A Type System for proving Depth Boundedness in the π-calculus

Emanuele D'Osualdo
University of Oxford, UK
emanuele.dosualdo@cs.ox.ac.uk

Luke Ong
University of Oxford, UK
lo@cs.ox.ac.uk

***Abstract*—The depth-bounded fragment of the π-calculus is an expressive class of systems enjoying decidability of some important verification problems. Unfortunately membership of the fragment is undecidable. We propose a novel type system, parameterised over a finite forest, that formalises name usage by π-terms in a manner that respects the forest. Type checking is decidable and type inference is computable; furthermore typable π-terms are guaranteed to be depth bounded.**

The second contribution of the paper is a proof of equivalence between the semantics of typable terms and nested data class memory automata, a class of automata over data words. We believe this connection can help to establish new links between the rich theory of infinite-alphabet automata and nominal calculi.

I. Introduction

The π-calculus [14] is a concise yet expressive model of concurrent computation. Its view of a concurrent system is a set of processes exchanging messages over channels, either private or public. Both processes and private channels can be created dynamically. A key feature of the calculus is mobility: a private channel name can be sent as a message over a public one and later used to exchange messages with an initially disconnected party. The *communication topology* of a π-calculus system, i.e., the graph linking processes that share channels, is therefore dynamically evolving, in contrast to those of simpler process calculi such as CCS.

From a verification point of view, proving properties of π-calculus terms is challenging: the full π-calculus is Turing-complete. As a consequence, a lot of research effort has been devoted to defining fragments of π-calculus that could be verified automatically while retaining as much expressivity as possible. To date, the most expressive fragment that has decidable verification problems is the *depth-bounded π-calculus* [8]. Roughly speaking, the depth of a π-calculus term can be understood as the maximum length of the simple (i.e non looping) paths in the communication topology of the term. A term is depth-bounded if there exists a $k \in \mathbb{N}$ such that the maximal nested depth of restriction of each reachable term is bounded by k. Notably, depth-bounded systems can have an infinite state-space and generate unboundedly many names. Besides enabling the design of procedures for deciding such important verification problems as termination or coverability, depth boundedness can be useful as a correctness property of a system in itself. Consider, for example, a system modelling an unbounded number of processes, each maintaining a private queue of tasks and communicating via message-passing. In the π-calculus, structures such as lists and queues are typically modelled using private channels to represent the "next" pointers. Proving a bound in depth k for such a system would guarantee that none of the queues grows unboundedly, which is an oft-desired resource-usage property.

Unfortunately, depth boundedness is a *semantic* property, it is undecidable whether a given arbitrary π-calculus term is depth-bounded. It has recently been proved that the problem becomes decidable if the bound k is fixed [18] but the complexity is very high.

Contributions

The first contribution of this paper is a novel fragment of π-calculus which we call *typably hierarchical*, which is a proper subset of the depth-bounded π-calculus. This fragment is defined by means of a type system with decidable checking and inference. The typably hierarchical fragment is rather expressive: it includes terms that are unbounded in the number of private channels and exhibit mobility.

The type system itself is based on the novel notion of *$\mathcal{T}$-compatibility*, where $\mathcal{T}$ is a given finite forest. We start from the observation that the communication topologies of depth bounded terms often exhibit a hierarchical structure: channels are organisable into layers with decreasing degree of sharing. Consider the example of an unbounded number of clients communicating with their local server: a message from a client containing a private channel is sent to the server's channel, the server replies to the client's request on the client's private channel. While the server's channel is shared among all the clients, the private channel of each client is shared only between itself and the server. $\mathcal{T}$-compatibility formalises and generalises this intuition. Roughly speaking, we associate to each channel name a *base type* which is a node in a (finite) forest $\mathcal{T}$. The forest $\mathcal{T}$ represents the hierarchical relationship between channels: it is the blueprint according to which one can organise the relationship between channels in each reachable term.

More precisely, the names hierarchy imposes constraints on the scopes of private names that can be considered valid. Consider the term $(\nu b.(\overline{a}\langle b\rangle.b(y))) \parallel a(x).(\nu c.\overline{x}\langle c\rangle)$: two parallel processes ready to synchronise on the public channel a. Upon synchronisation, the private name b—known only by the first process—will be transmitted to the second process

which will "migrate" under the scope of b. The result of this communication is the term $\nu b.(b(y) \parallel \nu c.(\overline{b}\langle c\rangle))$, note how the migration nests the scope of c in the scope of b. If $\mathcal{T}$ dictates that c is higher in the hierarchy than b the scoping resulting from the communication would be invalid: scope nesting should always respect the hierarchy. The type system we present constrains the use of names so that each term that is reachable from a typably hierarchical term is guaranteed to have scopes respecting $\mathcal{T}$. From this guarantee it can be shown that typably hierarchical terms have a depth bounded by the height of $\mathcal{T}$. We believe that the notion of $\mathcal{T}$-compatibility has potential as a specification device: it allows the user to specify the desired relationship between channels instead of just a numeric bound on depth.

After defining the typably hierarchical fragment, we turn to the question: *is there an automata-based model that can represent the same set of systems?* The second contribution of this paper is an encoding of typably hierarchical into *Nested Data Class Memory Automata* [3], a class of automata over data-words (i.e. finite words over infinite alphabets). An encoding of Nested Data Class Memory Automata into typably hierarchical terms is also presented, showing that the two models are equi-expressive. The two encodings are heavily based on the notion of $\mathcal{T}$-compatibility and open an approach to fruitful interactions between process algebra and automata over infinite alphabets.

II. Preliminaries

Labelled forests

A *forest* is a simple, acyclic, directed graph $f = (N_f, \prec_f)$ such that the edge relation, $\prec_f^{-1} \colon N_f \rightharpoonup N_f$, is the *parent* map which is defined on every node of the forest except the *root*(s). A *path* is a sequence of nodes, $n_1 \dots n_k$, such that for each $i < k$, $n_i \prec_f n_{i+1}$. Thus every node of a forest has a unique path to a root (and it follows that that root is unique). Henceforth we assume that all forests are finite. We write $\mathrm{paths}(f)$ for the set of paths in f. The *height* of a forest, $\mathrm{height}(f)$, is the length of its longest path.

An *L-labelled forest* is a pair $\varphi = (f_\varphi, \ell_\varphi)$ where f_φ is a forest and $\ell_\varphi \colon N_\varphi \to L$ is a labelling function on nodes. Given a path $n_1 \dots n_k$ of f_φ, its *trace* is the induced sequence $\ell_\varphi(n_1) \dots \ell_\varphi(n_k)$. By abuse of language, a *trace* is an element of L^* which is the trace of some path in the forest. We write $\mathrm{traces}(\varphi)$ for the set of traces of the labelled forest.

We define L-labelled forests inductively from the empty forest $(\emptyset, \emptyset)$. We write $\varphi_1 \uplus \varphi_2$ for the disjoint union of forests φ_1 and φ_2, and $l[\varphi]$ for the forest with a single root, labelled with $l \in L$, which has the respective roots of the forest φ as children. Since the choice of the set of nodes is irrelevant, we will always interpret equality between forests up to isomorphism (i.e. a bijection on nodes respecting parent and labeling).

The π-calculus

We use a π-calculus with guarded replication to express recursion [11]. Fix a universe $\mathcal{N}$ of names representing channels and messages occurring in communications. The syntax follows the grammar:

$$\begin{aligned} \mathcal{P} \ni P, Q ::= {} & \nu x.P \mid P_1 \parallel P_2 \mid M \mid M^* && \text{process} \\ M ::= {} & \mathbf{0} \mid M + M \mid \pi_i.P_i && \text{choice} \\ \pi ::= {} & a(x) \mid \overline{a}\langle b\rangle \mid \boldsymbol{\tau} && \text{prefix} \end{aligned}$$

Structural congruence is defined as the smallest congruence closed by α-conversion of bound names commutativity and associativity of choice and parallel composition with $\mathbf{0}$ as the neutral element, and the following laws for restriction, replication and scope extrusion:[1]

$$\nu x.\mathbf{0} \equiv \mathbf{0} \qquad \nu x.\nu y.P \equiv \nu y.\nu x.P \qquad \mathbf{0}^* \equiv \mathbf{0}$$

$$M^* \equiv M \parallel M^* \qquad P \parallel \nu a.Q \equiv \nu a.(P \parallel Q) \quad (\text{if } a \notin \mathrm{fn}(P))$$

The name x is bound in both $\nu x.P$, and in $a(x).P$. We will write $\mathrm{fn}(P)$, $\mathrm{bn}(P)$ and $\mathrm{bn}_\nu(P)$ for the set of free, bound and restriction-bound names in P, respectively. A sub-term is *active* if it is not under a prefix. A name is active when it is bound by an active restriction. The set $\mathrm{active}_\nu(P)$ is the set of the active names of P. Terms of the form M and M^* are called *sequential*. We write $\mathcal{S}$ for the set of all sequential terms. $\mathrm{seq}(P)$ is the set of all active sequential processes of P.

We will often rely on the following mild assumption, that the choice of names is unambiguous, especially when selecting a representative for a congruence class.

Name Uniqueness Assumption. Each name in P is bound at most once; and $\mathrm{fn}(P) \cap \mathrm{bn}(P) = \emptyset$.

Note that channels are unary; extending our work to the polyadic case is strightforward but we only consider the unary case for conciseness.

As we will see in the rest of the paper, the notions of depth and of hierarchy between names rely heavily on structural congruence. In particular, given a certain structure on names, there will be a specific representative of the structural congruence class that exhibits the desired properties. Nevertheless, we cannot assume the input term is always presented as that specific representative; worse yet, when the structure on names is not fixed, as in the case of type inference, we cannot fix any particular representative and be sure it will witness the desired properties. So, instead, in the semantics and in the type system, we manipulate a neutral representative called *normal form*, which is a variant of the *standard form* [13]. In this way we are not distracted by the particular syntactic representation we are presented with.

We say that a term P is in *normal form* ($P \in \mathcal{P}_{\mathsf{nf}}$) if it is in standard form and each of its inactive subterms is also in normal form. Formally, each process in normal form follows the grammar

$$\begin{aligned} \mathcal{P}_{\mathsf{nf}} \ni N ::= {} & \nu x_1.\cdots\nu x_n.(A_1 \parallel \cdots \parallel A_m) \\ A ::= {} & \pi_1.N_1 + \cdots + \pi_n.N_n \\ \mid {} & (\pi_1.N_1 + \cdots + \pi_n.N_n)^* \end{aligned}$$

[1] Technically, the $\mathbf{0}^* \equiv \mathbf{0}$ rule is not in the standard definition, but this does not affect the reduction semantics.

where the sequences $x_1 \dots x_n$ and $A_1 \dots A_m$ may be empty; when they are both empty the normal form is the term $\mathbf{0}$. We further assume w.l.o.g. that a normal form satisfies Name Uniqueness. Since the order of appearance of the restrictions, sequential terms or choices in a normal form is irrelevant in the technical development of our results, we use the following abbreviations. Given a finite set of indexes $I = \{i_1, \dots, i_n\}$ we write $\prod_{i \in I} A_i$ for $(A_{i_1} \parallel \cdots \parallel A_{i_n})$, which is $\mathbf{0}$ when I is empty; and $\sum_{i \in I} \pi_i.N_i$ for $(\pi_{i_1}.N_{i_1} + \cdots + \pi_{i_n}.N_{i_n})$. This notation is justified by commutativity and associativity of the parallel and choice operators. We also write $\nu X.P$ or $\nu x_1\, x_2 \cdots x_n.P$ for $\nu x_1.\cdots \nu x_n.P$ when $X = \{x_1, \dots, x_n\}$, or just P when X is empty; this is justified by the structural laws of restrictions. When X and Y are disjoint sets of names, we use juxtaposition for union.

Every process $P \in \mathcal{P}$ is structurally congruent to a process in normal form. The function $\mathrm{nf} \colon \mathcal{P} \to \mathcal{P}_{\mathsf{nf}}$, defined in Figure 1, extracts, from a term, a structurally equivalent normal form.

We are interested in the reduction semantics of a π-term, which can be described using the following rule.

Definition 1 (Semantics of π-calculus)**.** The operational semantics of π-calculus is defined by the transition system on π-terms, with transitions satisfying $P \to Q$ if

(i) $P \equiv \nu W.(S \parallel R \parallel C) \in \mathcal{P}_{\mathsf{nf}}$,
(ii) $S = (\overline{a}\langle b\rangle.\nu Y_s.S') + M_s$,
(iii) $R = (a(x).\nu Y_r.R') + M_r$,
(iv) $Q \equiv \nu W Y_s Y_r.(S' \parallel R'[\,b/x\,] \parallel C)$,

or if

(i) $P \equiv \nu W.(\boldsymbol{\tau}.\nu Y.P' \parallel C) \in \mathcal{P}_{\mathsf{nf}}$,
(ii) $Q \equiv \nu W Y.(P' \parallel C)$.

We define the set of reachable configurations as $\mathrm{Reach}(P) := \{\, Q \mid P \to^* Q \,\}$, writing $\to^*$ to mean the reflexive, transitive closure of $\to$.

Note that the use of structural congruence takes care of unfolding replications, if necessary.

Example 1 (Server/Client system). Consider the term $\nu s\, c.P$ where:

$$P = S^* \parallel C^* \parallel M^* \qquad S = s(x).\nu d.\overline{x}\langle d\rangle$$
$$C = c(m).(\overline{s}\langle m\rangle \parallel m(y).\overline{c}\langle m\rangle) \qquad M = \boldsymbol{\tau}.\nu m.\overline{c}\langle m\rangle$$

The term S^*, which is presented in normal form, represents a server listening to a port s for a client's requests. A request is a channel x that the client sends to the server for exchanging the response. After receiving x the server creates a new name d and sends it over x. The term M^* creates unboundedly many clients, each with its own private mailbox m. A client on a mailbox m repeatedly sends requests to the server and concurrently waits for the answer on the mailbox before recursing. An example run of the system:

$$\begin{aligned}\nu s\, c.P &\to \nu s\, c\, m.(P \parallel \overline{c}\langle m\rangle)\\ &\to \nu s\, c\, m.(P \parallel \overline{s}\langle m\rangle \parallel m(y).\overline{c}\langle m\rangle)\\ &\to \nu s\, c\, m\, d.(P \parallel \overline{m}\langle d\rangle \parallel m(y).\overline{c}\langle m\rangle)\\ &\to \nu s\, c\, m\, d.(P \parallel \overline{c}\langle m\rangle) \equiv \nu s\, c\, m.(P \parallel \overline{c}\langle m\rangle)\end{aligned}$$

Example 2 (Stack-like system). Consider the normal form $\nu X.(S^* \parallel \overline{s}\langle a\rangle)$ where $X = \{s, n, v, a\}$ and

$$S = s(x).\nu b.\big((\overline{v}\langle b\rangle.\overline{n}\langle x\rangle) \parallel \overline{s}\langle b\rangle\big)$$

The term $\overline{s}\langle a\rangle$ represents a stack with top element a; the stack is in an infinite loop that pushes new names (copies of b): this is represented by the term $\overline{v}\langle b\rangle.\overline{n}\langle a\rangle \parallel \overline{s}\langle b\rangle$ indicating that the top value is b, the next is a and the stack now starts from b. An example run:

$$\begin{aligned}&\nu X.(S^* \parallel \overline{s}\langle a\rangle)\\ &\to \nu X.(S^* \parallel \nu b.((\overline{v}\langle b\rangle.\overline{n}\langle a\rangle) \parallel \overline{s}\langle b\rangle))\\ &\to \nu X.(S^* \parallel \nu b\, b'.((\overline{v}\langle b\rangle.\overline{n}\langle a\rangle) \parallel (\overline{v}\langle b'\rangle.\overline{n}\langle b\rangle) \parallel \overline{s}\langle b'\rangle))\end{aligned}$$

The following definitions are minor variations of (but equivalent to) the concepts introduced in [8].[2]

Definition 2 (nest_ν, depth, depth-bounded term)**.** The *nesting of restrictions* of a term is given by the function

$$\begin{aligned}\mathrm{nest}_\nu(M) &:= \mathrm{nest}_\nu(M^*) := 0\\ \mathrm{nest}_\nu(\nu x.P) &:= 1 + \mathrm{nest}_\nu(P)\\ \mathrm{nest}_\nu(P \parallel Q) &:= \max(\mathrm{nest}_\nu(P), \mathrm{nest}_\nu(Q)).\end{aligned}$$

The *depth* of a term is defined as the minimal nesting of restrictions in its congruence class:

$$\mathrm{depth}(P) := \min\{\mathrm{nest}_\nu(Q) \mid P \equiv Q\}.$$

A term $P \in \mathcal{P}$ is *depth-bounded* if there exists a $k \in \mathbb{N}$ such that for each $Q \in \mathrm{Reach}(P)$, $\mathrm{depth}(Q) \le k$.

Example 3. The term in Example 1 is depth bounded: all the reachable terms are congruent to terms of the form

$$Q_{ijk} = \nu s\, c.\big(P \parallel N^i \parallel Req^j \parallel Ans^k\big)$$

for some $i, j, k \in \mathbb{N}$ where $N = \nu m.\overline{c}\langle m\rangle$, $Req = \nu m.(\overline{s}\langle m\rangle \parallel m(y).\overline{c}\langle m\rangle)$, $Ans = \nu m.(\nu d.\overline{m}\langle d\rangle \parallel m(y).\overline{c}\langle m\rangle)$ and by Q^n we mean the parallel composition of n copies of the term Q. For any i, j, k, $\mathrm{nest}_\nu(Q_{ijk}) \le 4$: the longest chain of nested restrictions is s, c, m, d.

The term in Example 2 is unbounded in depth: the number of nested copies of b grows every time a push is performed; it is not possible to extrude their scope to reduce the number of nested levels.

Note that both terms are not *name bounded* (in the sense of [6]): the number of active restrictions in the reachable terms is not bounded.

Definition 3 (Forest representation)**.** We represent the structural congruence class of a term $P \in \mathcal{P}$ with the set of

[2] In [8] these functions are defined on fragments. It is easy to prove that our definition of nest_ν coincides with the one in [8] on fragments and that for any fragment F and non-fragment P, if $F \equiv P$ then $\mathrm{nest}_\nu(P) \ge \mathrm{nest}_\nu(F)$. As a consequence our definition of depth coincides with the one in [8].

$$\mathrm{nf}(\mathbf{0}) := \mathbf{0} \qquad \mathrm{nf}(\pi.P) := \pi.\,\mathrm{nf}(P) \qquad \mathrm{nf}(\nu x.P) := \nu x.\,\mathrm{nf}(P)$$

$$\mathrm{nf}(M + M') := \begin{cases} \mathrm{nf}(M) & \text{if } \mathrm{nf}(M') = \mathbf{0} \neq \mathrm{nf}(M) \\ \mathrm{nf}(M') & \text{if } \mathrm{nf}(M) = \mathbf{0} \\ \mathrm{nf}(M) + \mathrm{nf}(M') & \text{otherwise} \end{cases} \qquad \mathrm{nf}(M^*) := \begin{cases} (\mathrm{nf}(M))^* & \text{if } \mathrm{nf}(M) \neq \mathbf{0} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$\mathrm{nf}(P \parallel Q) := \begin{cases} \mathrm{nf}(P) & \text{if } \mathrm{nf}(Q) = \mathbf{0} \neq \mathrm{nf}(P) \\ \mathrm{nf}(Q) & \text{if } \mathrm{nf}(P) = \mathbf{0} \\ \nu X_P X_Q.(N_P \parallel N_Q) & \text{if } \mathrm{nf}(Q) = \nu X_Q.N_Q, \mathrm{nf}(P) = \nu X_P.N_P \\ & \text{and } \mathrm{active}_\nu(N_P) = \mathrm{active}_\nu(N_Q) = \emptyset \end{cases}$$

Figure 1. Definition of the $\mathrm{nf} : \mathcal{P} \to \mathcal{P}_{\mathrm{nf}}$ function.

labelled forests $\mathcal{F}[\![P]\!] := \{\mathrm{forest}(Q) \mid Q \equiv P\}$ with labels in $\mathrm{active}_\nu(P) \uplus \mathrm{seq}(P)$ where $\mathrm{forest}(Q)$ is defined as

$$\mathrm{forest}(Q) := \begin{cases} x[\mathrm{forest}(Q')] & \text{if } Q = \nu x.Q' \\ \mathrm{forest}(Q_1) \uplus \mathrm{forest}(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ Q[(\emptyset, \emptyset)] & \text{if } Q \text{ is sequential} \\ (\emptyset, \emptyset) & \text{if } Q = \mathbf{0} \end{cases}$$

Note that only leaves are labelled with sequential processes.

The *restriction height*, $\mathrm{height}_\nu(\mathrm{forest}(P))$, is the length of the longest path formed of nodes labelled with names in $\mathrm{forest}(P)$.

Clearly, for any $P \in \mathcal{P}$, $\mathrm{depth}(P) = \min\{\mathrm{height}_\nu(\varphi) \mid \varphi \in \mathcal{F}[\![P]\!]\}$.

Lemma 1. *Let φ be a forest with labels in $\mathcal{N} \uplus \mathcal{S}$. Then $\varphi = \mathrm{forest}(Q)$ with $Q \equiv Q_\varphi$ where*

$$Q_\varphi := \nu X_\varphi.\textstyle\prod_{(n,A)\in I} A$$
$$X_\varphi := \{\ell_\varphi(n) \in \mathcal{N} \mid n \in N_\varphi\}$$
$$I := \{(n, A) \mid \ell_\varphi(n) = A \in \mathcal{S}\}$$

provided

1) *$\forall n \in N_\varphi$, if $\ell_\varphi(n) \in \mathcal{S}$ then n has no children in φ, and*
2) *$\forall n, n' \in N_\varphi$, if $\ell_\varphi(n) = \ell_\varphi(n') \in \mathcal{N}$ then $n = n'$, and*
3) *$\forall n \in N_\varphi$, if $\ell_\varphi(n) = A \in \mathcal{S}$ then for each $x \in X_\varphi \cap \mathrm{fn}(A)$ there exists $n' <_\varphi n$ such that $\ell_\varphi(n') = x$.*

Proof. We proceed by induction on the structure of φ. The base case is when $\varphi = (\emptyset, \emptyset)$, for which we have $Q_\varphi = \mathbf{0}$ and $\varphi = \mathrm{forest}(\mathbf{0})$.

When $\varphi = \varphi_0 \uplus \varphi_1$ we have that if conditions 1, 2 and 3 hold for φ, they must hold for φ_0 and φ_1 as well, hence we can apply the induction hypothesis to them obtaining $\varphi_i\, \mathrm{forest}(Q_i)$ with $Q_i \equiv Q_{\varphi_i}$ ($i \in \{0, 1\}$). We have $\varphi = \mathrm{forest}(Q_0 \parallel Q_1)$ by definition of forest, and we want to prove that $Q_0 \parallel Q_1 \equiv Q_\varphi$. By condition 2 on φ, X_{φ_0} and X_{φ_1} must be disjoint; furthermore, by condition 3 on both φ_0 and φ_1 we can conclude that $\mathrm{fn}(Q_{\varphi_i}) \cap X_{\varphi_{1-i}} = \emptyset$. We can therefore apply scope extrusion: $Q_0 \parallel Q_1 \equiv Q_{\varphi_0} \parallel Q_{\varphi_1} \equiv \nu X_{\varphi_0} X_{\varphi_1}.(P_{\varphi_0} \parallel P_{\varphi_1}) = Q_\varphi$.

The last case is when $\varphi = l[\varphi']$. Suppose conditions 1, 2 and 3 hold for φ. We distinguish two cases. If $l = A \in \mathcal{S}$, by 1 we have $\varphi' = (\emptyset, \emptyset)$, $\varphi = \mathrm{forest}(A)$ and $A = Q_\varphi$. If $l = x \in \mathcal{N}$ then we observe that conditions 1, 2 and 3 hold for φ' under the assumption that they hold for φ. Therefore $\varphi' = \mathrm{forest}(Q')$ with $Q' \equiv Q_{\varphi'}$, and, by definition of forest, $\varphi = \mathrm{forest}(\nu x.Q')$. By condition 2 we have $x \notin X_{\varphi'}$ so $\nu x.Q' \equiv \nu x.Q_{\varphi'} \equiv \nu(X \cup \{x\}).P_{\varphi'} = Q_\varphi$. □

III. The notion of $\mathcal{T}$-compatibility

In this section we will introduce the concept of *$\mathcal{T}$-compatibility*, which is a central tool in our constructions. First we will introduce types, which annotate names, and postulate that they are arranged as a forest $(\mathcal{T}, \prec)$. Intuitively, by annotating names with types we impose a hierarchy on them, and $\mathcal{T}$-compatibility of a term P will mean that the structure of P respects this hierarchy.

For the rest of the paper we will fix a *finite forest of base types* $(\mathcal{T}, \prec)$ where $n_1 \prec n_2$ means that "n_1 is the parent of n_2". We write $\leq$ and $<$ for the reflexive transitive and the transitive closure of $\prec$, respectively.

Types are of the form

$$\tau ::= t \mid t[\tau]$$

where $t \in \mathcal{T}$ is a base type. A name with type t cannot be used as a channel but can be used as a message; a name with type $t[\tau]$ can be used to transmit a name of type τ. We will write $\mathrm{base}(\tau)$ for t when $\tau = t[\tau']$ or $\tau = t$. Note that these are (a fragment of) the I/O-types in the sense of Pierce and Sangiorgi [16]. An environment Γ is a partial map from names to types, which we will write as a set of *type assignments*, $x : \tau$. Given a set of names X and an environment Γ, we write $\Gamma(X)$ for the set $\{\Gamma(x) \mid x \in X \cap \mathrm{dom}(\Gamma)\}$. Given two environments Γ and Γ' with $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$, we write $\Gamma\Gamma'$ for their union. For a type environment Γ we define $\min_{\mathcal{T}}(\Gamma) := \{(x : \tau) \in \Gamma \mid \forall (y : \tau') \in \Gamma.\ \mathrm{base}(\tau') \not< \mathrm{base}(\tau)\}$.

From now on, we will assume every π-term is annotated with types: in a restriction νX, X is a set of type assignments.

Definition 4 (Annotated term)**.** A *$\mathcal{T}$-annotated π-term* (or simply *annotated π-term*) $P \in \mathcal{P}^{\mathcal{T}}$ has the same syntax as

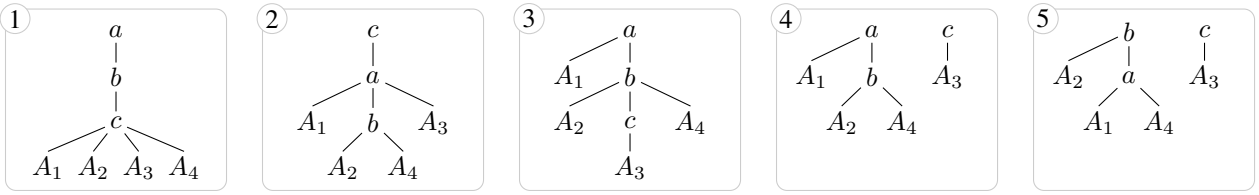


Figure 2. Examples of forests in $\mathcal{F}[\![P]\!]$ of Example 4: $P = \nu a\, b\, c.(A_1 \parallel A_2 \parallel A_3 \parallel A_4)$ where $A_1 = a(x)$, $A_2 = b(x)$, $A_3 = c(x)$ and $A_4 = \overline{a}\langle b\rangle$.

regular π-terms except restrictions take the form $\nu x : \tau$. The semantics is the same, except type annotations get copied when a name is duplicated or renamed by structural congruence. The definition of forest representation is also extended to annotated π-terms by changing the case when $Q = \nu x : \tau.Q'$ to $(x, t)[\text{forest}(Q')]$, where $\text{base}(\tau) = t$. The forests in $\mathcal{F}[\![P]\!]$ will thus have labels in $(\text{active}_\nu(P) \times \mathcal{T}) \uplus \text{seq}(P)$. We write $\mathcal{F}_\mathcal{T}$ for the set of forests with labels in $(\mathcal{N} \times \mathcal{T}) \uplus \mathcal{S}$. The set $\mathcal{P}^\mathcal{T}_{\mathsf{nf}}$ contains all the annotated π-terms in normal form.

Given a normal form $P = \nu X.\prod_{i\in I} A_i$ we say that A_i is *linked to* A_j *in* P, written $i \leftrightarrow_P j$, if $\text{fn}(A_i) \cap \text{fn}(A_j) \cap \{x \mid (x : \tau) \in X\} \neq \emptyset$. We also define the *tied-to* relation as the transitive closure of $\leftrightarrow_P$. I.e. A_i is *tied to* A_j, written $i \frown_P j$, if $\exists k \in I.\, i \leftrightarrow_P k \wedge k \frown_P j$. Furthermore, we say that a name y is *tied to* A_i *in* P, written $y \triangleleft_P i$, if $\exists j \in I.\, y \in \text{fn}(A_j) \wedge j \frown_P i$. Given an input-prefixed normal form $a(y).P$ where $P = \nu X.\prod_{i\in I} A_i$, we say that A_i *is migratable in* $a(y).P$, written $\text{Mig}_{a(y).P}(i)$, if $y \triangleleft_P i$.

The tied-to relation may seem obscure at first. Its meaning is better explained by the following lemma which indicates how this relation fundamentally constrains the possible shape of the forest of a term.

Lemma 2. *Let $P = \nu X.\prod_{i\in I} A_i \in \mathcal{P}^\mathcal{T}_{\mathsf{nf}}$, if $i \frown_P j$ then any forest $\varphi \in \mathcal{F}[\![P]\!]$ containing two leaves labelled with A_i and A_j respectively, will be such that these leaves belong to the same tree (i.e. have a common ancestor in φ).*

Proof. We show that the claim holds in the case where A_i is linked to A_j in P. From this, a simple induction over the length of linked-to steps required to prove $i \frown_P j$, can prove the lemma.

Suppose $i \leftrightarrow_P j$. Let $Y = \text{fn}(A_i) \cap \text{fn}(A_j) \cap \{x \mid (x : \tau) \in X\}$, we have $Y \neq \emptyset$. Both A_i and A_j are in the scope of each of the restrictions bounding names $y \in Y$ in any of the processes Q in the congruence class of P, hence, by definition of forest, the nodes labelled with A_i and A_j generated by $\text{forest}(Q)$ will have nodes labelled with $(y, \text{base}(X(y)))$ as common ancestors. □

Example 4. Take the normal form $P = \nu a\, b\, c.(A_1 \parallel A_2 \parallel A_3 \parallel A_4)$ where $A_1 = a(x)$, $A_2 = b(x)$, $A_3 = c(x)$ and $A_4 = \overline{a}\langle b\rangle$. We have $1 \leftrightarrow_P 4$, $2 \leftrightarrow_P 4$, therefore $1 \frown_P 2 \frown_P 4$ and $a \triangleleft_P 2$. In Figure 2 we show some of the forests in $\mathcal{F}[\![P]\!]$. Forest 1 represents $\text{forest}(P)$. The fact that A_1, A_2 and A_4 are tied is reflected by the fact that none of the forests place them in disjoint trees. Now suppose we select only the forests in $\mathcal{F}[\![P]\!]$ that have a as an ancestor of b: in all the forests in this set, the nodes labelled with A_1, A_2 and A_4 have a as common ancestor (as in forests 1, 2, 3 and 4). In particular, in these forests A_2 is necessarily a descendent of a even if a is not one of its free names.

Definition 5 ($\mathcal{T}$-compatibility)**.** Let $P \in \mathcal{P}^\mathcal{T}$ be an annotated π-term. A forest $\varphi \in \mathcal{F}[\![P]\!]$ is said to be $\mathcal{T}$*-compatible* if for every trace $((x_1, t_1) \ldots (x_k, t_k)\, A)$ in φ it holds that $t_1 < t_2 \ldots < t_k$. P is said to be $\mathcal{T}$*-compatible* if there exists a $\mathcal{T}$-compatible forest in $\mathcal{F}[\![P]\!]$. A term is $\mathcal{T}$*-shaped* if each of its subterms is $\mathcal{T}$-compatible.

Example 5. Let us fix $\mathcal{T}$ to be the forest $\mathsf{s} \prec \mathsf{c} \prec \mathsf{m} \prec \mathsf{d}$. The normal form in Example 1 is $\mathcal{T}$-compatible when s and c are annotated with types τ_s and τ_c respectively, with $\text{base}(\tau_s) = \mathsf{s}$ and $\text{base}(\tau_c) = \mathsf{c}$; indeed we have $\text{forest}(\nu(s : \tau_s)(c : \tau_c).P) = (s, \mathsf{s})\big[(c, \mathsf{c})[\, S^*[] \uplus C^*[] \uplus M^*[]\,]\big]$. By annotating m and d with types with base type m and d respectively, the term is also $\mathcal{T}$-shaped.

Since $\mathcal{T}$-compatibility is a condition on types, α-renaming does not interfere with it.

Lemma 3. *If $\text{forest}(P)$ is $\mathcal{T}$-compatible then for any term Q which is an α-renaming of P, $\text{forest}(Q)$ is $\mathcal{T}$-compatible.*

Lemma 4. *Let $P = \nu X.\prod_{i\in I} A_i$ be a $\mathcal{T}$-compatible normal form, $Y \subseteq X$ and $J \subseteq I$. Then $P' = \nu Y.\prod_{j\in J} A_j$ is $\mathcal{T}$-compatible.*

Proof. Take a $\mathcal{T}$-compatible forest $\varphi \in \mathcal{F}[\![P]\!]$. By Lemma 3 we can assume without loss of generality that $\varphi = \text{forest}(Q)$ where proving $Q \equiv P$ does not require α-renaming. Clearly, removing the leaves that do not correspond to sequential terms indexed by Y does not affect the $\mathcal{T}$-compatibility of φ. Similarly, if a restriction $(x : \tau) \in X$ is not in Y, we can remove the node of φ labelled with $(x, \text{base}(\tau))$ by making its parent the new parent of its children. This operation is unambiguous under Name Uniqueness and does not affect $\mathcal{T}$-compatibility, by transitivity of $<$. We then obtain a forest φ' which is $\mathcal{T}$-compatible and that, by Lemma 1, is the forest of a term congruent to the desired normal form P'. □

While many forests in $\mathcal{F}[\![P]\!]$ can be witnesses of the $\mathcal{T}$-compatibility of P, we want to characterise the shape of a witness that *must* exist if P is $\mathcal{T}$-compatible. Such forest is identified by $\Phi_\mathcal{T}(\text{nf}(P))$ where $\Phi_\mathcal{T} : \mathcal{P}^\mathcal{T}_{\mathsf{nf}} \to \mathcal{F}_\mathcal{T}$ is the function defined in Figure 3. We omit the subscript when irrelevant or clear from the context.

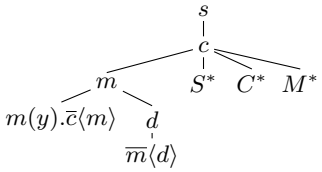
$$\Phi_{\mathcal{T}}(\nu X.\textstyle\prod_{i\in I} A_i) := \begin{cases} \biguplus_{i\in I}\{A_i[]\} & \text{if } X = \emptyset \\ \left(\biguplus\{(x,\mathrm{base}(\tau))[\Phi_{\mathcal{T}}(\nu Y_x.\prod_{j\in I_x} A_j)] \mid (x:\tau)\in \min_{\mathcal{T}}(X)\}\right) \uplus \Phi_{\mathcal{T}}(\nu Z.\prod_{r\in R} A_r) & \text{if } X \neq \emptyset \end{cases}$$

where $P = \nu X.\prod_{i\in I} A_i$ and

$$I_x = \{i \in I \mid x \lhd_P i\} \qquad R = I \setminus (\textstyle\bigcup_{(x:\tau)\in\min_{\mathcal{T}}(X)} I_x)$$
$$Y_x = \{(y:\tau) \in X \mid \exists i \in I_x \,.\, y \in \mathrm{fn}(A_i)\} \setminus \min_{\mathcal{T}}(X) \qquad Z = X \setminus (\textstyle\bigcup_{(x:\tau)\in\min_{\mathcal{T}}(X)} Y_x \cup \{x:\tau\})$$

Figure 3. Definition of $\Phi_{\mathcal{T}} \colon \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}} \to \mathcal{F}_{\mathcal{T}}$.

Example 6. In the run shown in Example 1, after three steps we reach $Q = \nu s\, c\, m\, d.(P \parallel \overline{m}\langle d\rangle \parallel m(y).\overline{c}\langle m\rangle)$. The forest $\Phi_{\mathcal{T}}(Q)$, when $\mathcal{T}$ and types annotations are as in Example 5, is



where the nodes show only the name components of their labels for conciseness. Note how the scope of names is minimised while respecting $\mathcal{T}$-compatibility.

Consider the term P in Example 4, with annotations $a:\mathsf{a}[\mathsf{b}[t]]$, $b:\mathsf{b}[t]$ and $c:\mathsf{c}[t']$. Forests 4 and 5 of Figure 2 represent $\Phi_{\mathcal{T}}(P)$ when $\mathcal{T}$ is $\mathsf{a} \prec \mathsf{b}$ and $\mathsf{b} \prec \mathsf{a}$ respectively.

Lemma 5. *Let $P \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$. Then:*

a) $\Phi_{\mathcal{T}}(P)$ is a $\mathcal{T}$-compatible forest;
b) $\Phi_{\mathcal{T}}(P) \in \mathcal{F}[\![P]\!]$ if and only if P is $\mathcal{T}$-compatible;
c) if $P \equiv Q \in \mathcal{P}^{\mathcal{T}}$ then $\Phi_{\mathcal{T}}(P) \in \mathcal{F}[\![Q]\!]$ if and only if Q is $\mathcal{T}$-compatible.

Proof. Item a) is an easy induction on the cardinality of X.

Item b) requires more work. By item a) $\Phi(P)$ is $\mathcal{T}$-compatible so $\Phi(P) \in \mathcal{F}[\![P]\!]$ proves that P is $\mathcal{T}$-compatible.

To prove the $\Leftarrow$-direction we assume that $P = \nu X.\prod_{i\in I} A_i$ is $\mathcal{T}$-compatible and proceed by induction on the cardinality of X to show that $\Phi(P) \in \mathcal{F}[\![P]\!]$. The base case is when $X = \emptyset$: $\Phi(P) = \Phi(\prod_{i\in I} A_i) = \biguplus_{i\in I}\{A_i[]\} = \mathrm{forest}(\prod_{i\in I} A_i) = \mathrm{forest}(P) \in \mathcal{F}[\![P]\!]$. For the induction step, we observe that $X \neq \emptyset$ implies $\min_{\mathcal{T}}(X) \neq \emptyset$ so, $Z \subset X$ and for each $(x:\tau) \in \min_{\mathcal{T}}(X)$, $Y_x \subset X$ since $x \notin Y_x$. This, together with Lemma 4, allows us to apply the induction hypotesis on the terms $P_x = \nu Y_x.\prod_{j\in I_x} A_j$ and $P_R = \nu Z.\prod_{r\in R} A_r$, obtaining that there exist terms $Q_x \equiv P_x$ and $Q_R \equiv P_R$ such that $\mathrm{forest}(Q_x) = \Phi(P_x)$ and $\mathrm{forest}(Q_R) = \Phi(P_R)$ where all the forests $\mathrm{forest}(Q_x)$ and $\mathrm{forest}(Q_R)$ are $\mathcal{T}$-compatible. Let $Q = \prod\{\nu(x:\tau).Q_x \mid (x:\tau) \in \min_{\mathcal{T}}(X)\} \parallel Q_R$, then $\mathrm{forest}(Q) = \Phi(P)$. To prove the claim we only need to show that $Q \equiv P$. We have $Q \equiv \prod\{\nu(x:\tau).\nu Y_x.\prod_{j\in I_x} A_j \mid (x:\tau) \in \min_{\mathcal{T}}(X)\} \parallel P_R$ and we want to apply extrusion to get $Q \equiv \nu Y_{\min}.(\prod_{i\in I_{\min}} A_i) \parallel P_R$ for $I_{\min} = \biguplus\{I_x \mid (x:\tau) \in \min_{\mathcal{T}}(X)\}$, $Y_{\min} = \min_{\mathcal{T}}(X) \uplus \biguplus\{Y_x \mid (x:\tau) \in \min_{\mathcal{T}}(X)\}$ which adds an obligation to prove that

i) I_x are all pairwise disjoint so that $I_{\min}$ is well-defined,
ii) Y_x are all pairwise disjoint and all disjoint from $\min_{\mathcal{T}}(X)$ so that $Y_{\min}$ is well-defined,
iii) $Y_x \cap \mathrm{fn}(A_j) = \emptyset$ for every $j \in I_z$ with $z \neq x$ so that we can apply the extrusion rule.

To prove condition i), assume by contradiction that there exists an $i \in I$ and names $x, y \in \min_{\mathcal{T}}(X)$ with $x \neq y$, such that both x and y are tied to A_i in P. By transitivity of the tied-to relation, we have $I_x = I_y$. By Lemma 2 all the A_j with $j \in I_x$ need to be in the same tree in any forest $\varphi \in \mathcal{F}[\![P]\!]$. Since P is $\mathcal{T}$-compatible there exist such a φ which is $\mathcal{T}$-compatible and has every A_j as label of leaves of the same tree. This tree will include a node n_x labelled with $(x, \mathrm{base}(X(x)))$ and a node n_y labelled with $(y, \mathrm{base}(X(y)))$. By $\mathcal{T}$-compatibility of φ and the existence of a path between n_x and n_y we infer $\mathrm{base}(X(x)) < \mathrm{base}(X(y))$ or $\mathrm{base}(X(y)) < \mathrm{base}(X(x))$ which contradicts the assumption that $x, y \in \min_{\mathcal{T}}(X)$.

Condition ii) follows from condition i): suppose there exists a $(z:\tau) \in X \cap Y_x \cap Y_y$ for $x \neq y$, then we would have that $z \in \mathrm{fn}(A_i) \cap \mathrm{fn}(A_j)$ for some $i \in I_x$ and $j \in I_y$, but then $i \frown_P j$, meaning that $i \in I_y$ and $j \in I_x$ violating condition i). The fact that $Y_x \cap \min_{\mathcal{T}}(X) = \emptyset$ follows from the definition of Y_x. The same reasoning proves condition iii).

Now we have $Q \equiv \nu Y_{\min}.(\prod_{i\in I_{\min}} A_i) \parallel \nu Z.\prod_{r\in R} A_r$ and we want to apply extrusion again to get $Q \equiv \nu Y_{\min} Z.\prod\{A_i \mid i \in (I_{\min} \uplus R)\}$ which is sound under the following conditions:

iv) $Y_{\min} \cap Z = \emptyset$,
v) $I_{\min} \cap R = \emptyset$,
vi) $Z \cap \mathrm{fn}(A_i) = \emptyset$ for all $i \notin R$

of which the first two hold trivially by construction, while the last follows from condition viii) below, as a name in the intersection of Z and a $\mathrm{fn}(A_i)$ would need to be in X but not in $Y_{\min}$. To be able to conclude that $Q \equiv P$ it remains to prove that

vii) $I = I_{\min} \uplus R$ and
viii) $X = Y_{\min} \uplus Z$

which are also trivially valid by inspection of their definitions. This concludes the proof for item b).

Finally, for every $Q \in \mathcal{P}^{\mathcal{T}}$ such that $Q \equiv P$, $\Phi(P) \in \mathcal{F}[\![Q]\!]$ if and only if $\Phi(P) \in \mathcal{F}[\![P]\!]$ by definition of $\mathcal{F}[\![-]\!]$; since $\Phi(P)$ is $\mathcal{T}$-compatible we can infer that Q is $\mathcal{T}$-compatible if and only if $\Phi(P) \in \mathcal{F}[\![Q]\!]$, which proves item c). □

Lemma 6. *Let $P = \nu X.\prod_{i\in I} A_i \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ be a $\mathcal{T}$-compatible normal form. Then for every trace $((x_1,t_1)\dots(x_k,t_k)\,A_j)$ in*

the forest $\Phi(P)$, for every $i \in \{1,\dots,k\}$, we have $x_i \lhd_P j$ (i.e. x_i is tied to A_j in P).

Proof. Straightforward from the definition of I_x in Φ: when a node labelled by (x,t) is introduced, its subtree is extracted from a recursive call on a term that contains all and only the sequential terms that are tied to x. □

Remark 1. *$\Phi(P)$ satisfies conditions 1, 2 and 3 of Lemma 1.*

It is clear from the definition that if a π-term P is $\mathcal{T}$-compatible then $\mathrm{depth}(P)$ is bounded by the length of the longest strictly increasing chain in $\mathcal{T}$; since $\mathcal{T}$ is assumed to be finite, the bound on the depth is finite.

Proposition 1. *Let $\mathcal{T}$ be a forest and P an annotated π-term. If every $Q \in \mathrm{Reach}(P)$ is $\mathcal{T}$-compatible, then P is depth-bounded.*

Example 7. Fix $\mathcal{T}$ to be the forest $\mathsf{n} \prec \mathsf{v} \prec \mathsf{s} \prec \mathsf{a}$ and take the term of Example 2 annotating it with types such that the base types of the names n, v, s, a and b are $\mathsf{n}, \mathsf{v}, \mathsf{s}, \mathsf{a}$ and a respectively. The term $\nu n\, v\, s\, a.(S^* \parallel \overline{s}\langle a\rangle)$ is $\mathcal{T}$-compatible, but the term $Q = \nu nvsabb'.(S^* \parallel (\overline{v}\langle b\rangle.\overline{n}\langle a\rangle) \parallel (\overline{v}\langle b'\rangle.\overline{n}\langle b\rangle) \parallel \overline{s}\langle b'\rangle)$, reachable from it, is not: b and b' have the same base type a but need to be in the same trace in any forest of $\mathcal{F}[\![Q]\!]$. As we have shown in Example 3, this term is not bounded in depth, so there cannot be any finite $\mathcal{T}$ such that every reachable term is $\mathcal{T}$-compatible.

IV. A TYPE SYSTEM FOR HIERARCHICAL TOPOLOGIES

We now define a type system to prove depth boundedness. Our goal is to use Proposition 1 by devising a type system, parametrised over $\mathcal{T}$, such that typability implies invariance of $\mathcal{T}$-compatibility under reduction. Typability of a $\mathcal{T}$-compatible term P would then imply that every term reachable from it is $\mathcal{T}$-compatible, entailing depth boundedness of P.

A judgement $\Gamma \vdash_{\mathcal{T}} P$ means that $P \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ can be typed under assumptions Γ, over the tree $\mathcal{T}$; we say that P is *typable* if $\Gamma \vdash_{\mathcal{T}} P$ is provable for some Γ and $\mathcal{T}$. An arbitrary term $P \in \mathcal{P}^{\mathcal{T}}$ is said to be *typable* if its normal form is. The typing rules are presented in Figure 4.

The type system presents several non-standard features. First, it is defined on normal forms as opposed to general π-terms. This choice is motivated by the fact that different syntactic presentations of the same term may be misleading when trying to analyse the relation between the structure of the term and $\mathcal{T}$. The rules need to guarantee that a reduction will not break $\mathcal{T}$-compatibility, which is a property of the congruence class of the term. As justified by Lemma 2, the scope of names in a congruence class may vary, but the tied-to relation puts constraints on the structure that must be obeyed by all members of the class. Therefore the type system is designed around this basic concept, rather than the specific scoping of any representative of the structural congruence class. Second, no type information is associated with the typed term, only restricted names hold type annotations. Third, while the rules are compositional, the constraints on base types have a global flavour due to the fact that they involve the structure of $\mathcal{T}$ which is a global parameter of typing proofs.

Let us illustrate intuitively how the constraints enforced by the rules guarantee preservation of $\mathcal{T}$-compatibility. Consider the term

$$P = \nu e\, a.\Big(\nu b.\big(\overline{a}\langle b\rangle.A_0\big) \parallel \nu d.\big(a(x).A\big)\Big)$$

with $A = \nu c.(A_1 \parallel A_2 \parallel A_3)$, $A_0 = b(y)$, $A_1 = \overline{x}\langle c\rangle$, $A_2 = c(z).\overline{a}\langle e\rangle$ and $A_3 = \overline{a}\langle d\rangle$. Let $\mathcal{T}$ be the forest with $t_e \prec t_a \prec t_b \prec t_c$ and $t_a \prec t_d$, where t_x is the base type of the (omitted) annotation of the restriction νx, for $x \in \{a,b,c,d,e\}$. The reader can check that $\mathrm{forest}(P)$ is $\mathcal{T}$-compatible. In the traditional understanding of mobility, we would interpret the communication of b over x as an application of scope extrusion to include $\nu d.\big(a(x).A\big)$ in the scope of b and then syncronisation over a with the application of the substitution $[\,b/x\,]$ to A; note that the substitution is only valid because the scope of b has been extended to include the receiver. Our key observation is that we can instead interpret this communication as a migration of the subcomponents of A that do get their scopes changed by the reduction, from the scope of the receiver to the scope of the sender. For this operation to be sound, the subcomponents of A migrating to the sender's scope cannot use the names that are in the scope of the receiver but not of the sender. In our specific example, after the synchronisation between the prefixes $\overline{a}\langle b\rangle$ and $a(x)$, b is substituted to x in A_1 resulting in the term $A_1' = \overline{b}\langle c\rangle$ and A_0, A_1', A_2 and A_3 become active. The scope of A_0 can remain unchanged as it cannot know more names than before as a result of the communication. By contrast, A_1 now knows b as a result of the substitution $[\,b/x\,]$: A_1 needs to migrate under the scope of b. Since A_1 uses c as well, the scope of c needs to be moved under b; however A_2 uses c so it needs to migrate under b with the scope of c. A_3 instead does not use neither b nor c so it can avoid migration and its scope remains unaltered. This information can be formalised using the tied-to relation: on one hand, A_1 and A_2 need to be moved together because $1 \frown_A 2$ and they need to be moved because $x \lhd_{a(x).A} 1, 2$. On the other hand, A_3 is not tied to neither A_1 nor A_2 in A and does not know x, thus it is not migratable. After reduction, our view of the reactum is the term

$$\nu a.\Big(\nu b.\big(A_0 \parallel \nu c.(A_1' \parallel A_2)\big) \parallel \nu d.A_3\Big)$$

the forest of which is $\mathcal{T}$-compatible. Rule PAR, applied to A_1 and A_2, ensures that c has a base type that can be nested under the one of b. Rule IN does not impose constraints on the base types of A_3 because A_3 is not migratable. It does however check that the base type of e is an ancestor of the one of a, thus ensuring that both receiver and sender are already in the scope of e. The base type of a does not need to be further constrained since the fact that the synchronisation happened on it implies that both the receiver and the sender were already under its scope; this implies, by $\mathcal{T}$-compatibility of P, that c can be nested under a.

We now describe the purpose of the rules of the type system in more detail. Most of the rules just drive the derivation

$$\frac{\forall i \in I.\, \Gamma, X \vdash_{\mathcal{T}} A_i \qquad \forall i \in I.\, \forall x{:}\tau_x \in X.\, x \lhd_P i \implies \mathrm{base}(\Gamma(\mathrm{fn}(A_i))) < \mathrm{base}(\tau_x)}{\Gamma \vdash_{\mathcal{T}} \nu X.\prod_{i\in I} A_i}\ \textsc{Par} \qquad \frac{\forall i \in I.\, \Gamma \vdash_{\mathcal{T}} \pi_i.P_i}{\Gamma \vdash_{\mathcal{T}} \sum_{i\in I} \pi_i.P_i}\ \textsc{Choice}$$

$$\frac{\Gamma \vdash_{\mathcal{T}} A}{\Gamma \vdash_{\mathcal{T}} A^*}\ \textsc{Repl} \qquad \frac{\Gamma \vdash_{\mathcal{T}} A}{\Gamma \vdash_{\mathcal{T}} \boldsymbol{\tau}.A}\ \textsc{Tau} \qquad \frac{a{:}t_a[\tau_b] \in \Gamma \qquad b{:}\tau_b \in \Gamma \qquad \Gamma \vdash_{\mathcal{T}} Q}{\Gamma \vdash_{\mathcal{T}} \overline{a}\langle b\rangle.Q}\ \textsc{Out}$$

$$\frac{a{:}t_a[\tau_x] \in \Gamma \qquad \Gamma, x{:}\tau_x \vdash_{\mathcal{T}} P \qquad \mathrm{base}(\tau_x) \le t_a \vee \left(\forall i \in I.\ \mathrm{Mig}_{a(x).P}(i) \implies \mathrm{base}(\Gamma(\mathrm{fn}(A_i) \setminus \{a\})) < t_a\right)}{\Gamma \vdash_{\mathcal{T}} a(x).\nu X.\prod_{i\in I} A_i}\ \textsc{In}$$

Figure 4. A type system for proving depth boundedness. The term P stands for $\nu X.\prod_{i\in I} A_i$.

through the structure of the term. The crucial constraints are checked by Par, In and Out.

The Out rule is the one enforcing types to be consistent with the dataflow of the process: the type of the argument of a channel a must agree with the types of all the names that may be sent over a. This is a very coarse sound over-approximation of the dataflow; if necessary it could be refined using well-known techniques from the literature but a simple approach is sufficient here to type interesting processes.

Rule Par is best understood imagining the normal form to be typed, P, as the continuation of a prefix $\pi.P$. In this context a reduction exposes each of the active sequential subterms of P which need to have a place in a $\mathcal{T}$-compatible forest for the reactum. The constraint in Par can be read as follows. A "new" leaf A_i may refer to names already present in the forests of the reaction context; these names are the ones mentioned in both $\mathrm{fn}(A_i)$ and Γ. Then we must be able to insert A_i so that we can find these names in its path. However, A_i must belong to a tree containing all the names in X that are tied to it in P. So by requiring every name tied to A_i to have a base type smaller than any name in the context that A_i may refer to, we make sure that we can insert the continuation in the forest of the context without violating $\mathcal{T}$-compatibility. Note that $\Gamma(\mathrm{fn}(A_i))$ contains only types that annotate names both in Γ and $\mathrm{fn}(A_i)$, that is, names which are not restricted by X and are referenced by A_i (and therefore come from the context).

Rule In serves two purposes: on the one hand it requires the type of the messages that can be sent through a to be consistent with the use of the variable x which will be bound to the messages; on the other hand, it constrains the base types of a and x so that synchronisation can be performed without breaking $\mathcal{T}$-compatibility. The second purpose is achieved by distinguishing two cases, represented by the two disjuncts of the condition on base types of the rule. In the first case the base type of the message is an ancestor of the base type of a in $\mathcal{T}$. This implies that in any $\mathcal{T}$-compatible forest representing $a(x).P$, the name b sent as message over a is already in the scope of P. Under this circumstance, there is no real migration and the substitution $[\,b/x\,]$ does not alter the scope of P and the $\mathcal{T}$-compatibility constraints to be satisfied are in essence unaltered. The second case is more complicated as it involves migration. This case also requires a slightly non-standard feature: the premises predicate not only on the direct subcomponents of an input prefixed term, but also on the direct subcomponents of the continuation. This is needed to be able to separate the continuation in two parts: the one requiring migration and the one that does not. The non migratable sequential terms behave exactly as the case of the first disjunct: their scope is unaltered. The migratable ones instead are intended to be inserted as descendent of the node representing the message in the forest of the reaction context. For this to be valid without rearrangement of the forest of the context, we need all the names in the context that are referenced in the migratable terms, to be already in their scope; we make sure this is the case by requiring the free names of any migratable A_i that are from the context (i.e. in Γ) to have base types smaller than the base type of a. The set $\mathrm{base}(\Gamma(\mathrm{fn}(A_i) \setminus \{a\}))$ indeed represents the base types of the names in the reaction context referenced in a migratable continuation A_i. In fact a is a name that needs to be in the scope of both the sender and the receiver at the same time, so it needs to be a common ancestor of sender and receiver in any $\mathcal{T}$-compatible forest. Any name in the reaction context and in the continuation of the receiver, with a base type smaller than the one of a, will be an ancestor of a—and hence of the sender, the receiver and the node representing the message—in any $\mathcal{T}$-compatible forest. Clearly, remembering a is not harmful as it must be already in the scope of receiver and sender so we exclude it from the constraint.

Example 8. Take the normal form in Example 1. Let us fix $\mathcal{T}$ to be the forest $\mathsf{s} \prec \mathsf{c} \prec \mathsf{m} \prec \mathsf{d}$ and annotate the normal form with the following types: $s{:}\tau_s = \mathsf{s}[\tau_m]$, $c{:}\tau_c = \mathsf{c}[\tau_m]$, $m{:}\tau_m = \mathsf{m}[\mathsf{d}]$ and $d{:}\mathsf{d}$. Let $\Gamma = \{(s{:}\tau_s), (c{:}\tau_c)\}$. We want to prove $\emptyset \vdash_{\mathcal{T}} \nu sc.P$. We can apply rule Par: in this case there are no conditions on types because, being the environment empty, we have $\mathrm{base}(\emptyset(\mathrm{fn}(A))) = \emptyset$ for every active sequential term A of P. The rule requires $\Gamma \vdash_{\mathcal{T}} S^*$, $\Gamma \vdash_{\mathcal{T}} C^*$ and $\Gamma \vdash_{\mathcal{T}} M^*$, which can be proved by proving typability of S, C and M under Γ by rule Repl. To prove $\Gamma \vdash_{\mathcal{T}} S$ we apply rule In; we have $s{:}\mathsf{s}[\tau_m] \in \Gamma$ and we need to prove that $\Gamma, x{:}\tau_m \vdash_{\mathcal{T}} \nu d.\overline{x}\langle d\rangle$. No constraints on base types are generated at this step since the migratable sequential term $\nu d.\overline{x}\langle d\rangle$ does not contain free variables typed by Γ making $\Gamma(\mathrm{fn}(\nu d.\overline{x}\langle d\rangle) \setminus \{a\}) = \Gamma(\{x\})$ empty. Next, $\Gamma, x{:}\tau_m \vdash_{\mathcal{T}} \nu d.\overline{x}\langle d\rangle$ can be proved by applying rule Par which amounts to checking $\Gamma, x{:}\tau_m \vdash_{\mathcal{T}} \overline{x}\langle d\rangle.\mathbf{0}$ (by a simple application of Out and the axiom $\Gamma, x{:}\tau_m \vdash_{\mathcal{T}} \mathbf{0}$) and

verifying the condition—true in $\mathcal{T}$—$\text{base}(\tau_m) < \text{base}(\tau_d)$: in fact d is tied to $\overline{x}\langle d\rangle$ and, for $\Gamma' = \Gamma \cup \{x:\tau_m\}$, $\text{base}(\Gamma'(\text{fn}(\overline{x}\langle d\rangle))) = \text{base}(\Gamma'(\{x,d\})) = \text{base}(\{\tau_m\})$. The proof for $\Gamma \vdash_{\mathcal{T}} M$ is similar and requires $\mathsf{c} < \mathsf{m}$ which is true in $\mathcal{T}$. Finally, we can proof $\Gamma \vdash_{\mathcal{T}} C$ using rule IN; both the two continuation $A_1 = \overline{s}\langle m\rangle$ and $A_2 = m(y).\overline{c}\langle m\rangle$ are migratable in C and since $\text{base}(\tau_m) < \text{base}(\tau_c)$ is false we need the other disjunct of the condition to be true. This amounts to check that $\text{base}(\Gamma(\text{fn}(A_1) \setminus \{c\})) = \text{base}(\Gamma(\{s,m\})) = \text{base}(\{\tau_s\}) < \mathsf{c}$ (note $m \notin \text{dom}(\Gamma)$) and $\text{base}(\Gamma(\text{fn}(A_a) \setminus \{c\})) = \text{base}(\Gamma(\emptyset)) < \mathsf{c}$ (that holds trivially). Fortunately, this is the case in $\mathcal{T}$. To complete the typing we need to show $\Gamma, m:\tau_m \vdash_{\mathcal{T}} A_1$ and $\Gamma, m:\tau_m \vdash_{\mathcal{T}} A_2$. The former can be proved by a simple application of OUT which does not impose further constraints on $\mathcal{T}$. The latter is proved by applying IN which requires $\text{base}(\tau_c) < \mathsf{m}$, which holds in $\mathcal{T}$. Note how, at every step, there is only one rule that applies to each subproof.

Example 9. There is no choice for (a finite) $\mathcal{T}$ that would make the normal form in Example 2 typeable. To see why, one can build the proof tree without assumptions on $\mathcal{T}$ obtaining that:

1) the restrictions must be annotated with types consistent with the type assignments
$$s:t_s[t] \qquad v:t_v[t] \qquad n:t_n[t] \qquad a:t \qquad b:t$$
2) $\mathcal{T}$ must satisfy the constraint that the base type assigned to b must be strictly greater than the one assigned to x, which is inconsistent with $s:t_s[t], b:t$.

A. Soundness

In this section we show how the type system can be used to prove depth-boundedness. Theorem 1 will show how typability is preserved by reduction. Theorem 2 establishes the main property of the type system: if a term is typable then $\mathcal{T}$-shapedness is invariant under reduction. This allows us to conclude that if a term is $\mathcal{T}$-shaped and typable, then every term reachable from it will be $\mathcal{T}$-shaped and, therefore, it is depth-bounded.

We start with some simple properties of the type system.

Lemma 7. *Let $P \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ and Γ, Γ' be type environments.*

a) if $\Gamma \vdash_{\mathcal{T}} P$ then $\text{fn}(P) \subseteq \text{dom}(\Gamma)$;
b) if $\text{dom}(\Gamma') \cap \text{bn}(P) = \emptyset$ and $\text{fn}(P) \subseteq \text{dom}(\Gamma)$, then $\Gamma \vdash_{\mathcal{T}} P$ if and only if $\Gamma\Gamma' \vdash_{\mathcal{T}} P$;
c) if $P \equiv P' \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ then, $\Gamma \vdash_{\mathcal{T}} P$ if and only if $\Gamma \vdash_{\mathcal{T}} P'$.

The subtitution lemma states that substituting names without altering the types preserves typability.

Lemma 8 (Substitution)**.** *Let $P \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ and Γ be a typing environment including the type assignments $a:\tau$ and $b:\tau$. Then it holds that if $\Gamma \vdash_{\mathcal{T}} P$ then $\Gamma \vdash_{\mathcal{T}} P[b/a]$.*

Proof. We prove the lemma by induction on the structure of P. The base case is when $P \equiv \mathbf{0}$, where the claim trivially holds.

For the induction step, let $P \equiv \nu X.\prod_{i\in I} A_i$ with $A_i = \sum_{j\in J} \pi_{ij}.P_{ij}$, for some finite sets of indexes I and J. Since the presence of replication does not affect the typing proof, we can safely ignore that case as it follows the same argument. Let us assume $\Gamma \vdash_{\mathcal{T}} P$ and prove that $\Gamma \vdash_{\mathcal{T}} P[b/a]$.

Let Γ' be $\Gamma \cup X$. From $\Gamma \vdash_{\mathcal{T}} P$ we have

$$\Gamma, X \vdash_{\mathcal{T}} A_i \tag{1}$$
$$x \triangleleft_P i \implies \text{base}(\Gamma(\text{fn}(A_i))) < \text{base}(\tau_x) \tag{2}$$

for each $i \in I$ and $x:\tau_x \in X$. To extract from this assumptions a proof for $\Gamma \vdash_{\mathcal{T}} P[b/a]$, we need to prove that (1) and (2) hold after the substitution.

Since the substitution does not apply to names in X and the *tied to* relation is only concerned with names in X, the only relevant effect of the substitution is modifying the set $\text{fn}(A_i)$ to $\text{fn}(A_i[b/a]) = \text{fn}(A_i) \setminus \{a\} \cup \{b\}$ when $a \in \text{fn}(A_i)$; But since $\Gamma(a) = \Gamma(b)$ by hypothesis, we have $\text{base}(\Gamma(\text{fn}(A_i[b/a]))) < \text{base}(\tau_x)$.

It remains to prove (1) holds after the substitution as well. This amounts to prove for each $j \in J$ that $\Gamma' \vdash_{\mathcal{T}} \pi_{ij}.P_{ij} \implies \Gamma' \vdash_{\mathcal{T}} \pi_{ij}.P_{ij}[b/a]$; we prove this by cases.

Suppose $\pi_{ij} = \overline{\alpha}\langle\beta\rangle$ for two names α and β, then from $\Gamma' \vdash_{\mathcal{T}} \pi_{ij}.P_{ij}$ we know the following

$$\alpha:t_\alpha[\tau_\beta] \in \Gamma' \qquad \beta:\tau_\beta \in \Gamma' \tag{3}$$
$$\Gamma' \vdash_{\mathcal{T}} P_{ij} \tag{4}$$

Condition (3) is preserved after the substitution because it involves only types so, even if α or β are a, their types will be left untouched after they get substituted with b from the hypothesis that $\Gamma(a) = \Gamma(b)$. Condition (4) implies $\Gamma' \vdash_{\mathcal{T}} P_{ij}[b/a]$ by inductive hypothesis.

Suppose now that $\pi_{ij} = \alpha(x)$ and $P_{ij} \equiv \nu Y.\prod_{k\in K} A'_k$ for some finite set of indexes K; by hypothesis we have:

$$\alpha:t_\alpha[\tau_x] \in \Gamma' \tag{5}$$
$$\Gamma', x:\tau_x \vdash_{\mathcal{T}} P_{ij} \tag{6}$$
$$\text{base}(\tau_x) \leq t_\alpha \vee \forall k \in K.\ \text{Mig}_{\pi_{ij}.P_{ij}}(k) \implies \text{base}(\Gamma'(\text{fn}(A'_k) \setminus \{\alpha\})) < t_\alpha \tag{7}$$

Now x and Y are bound names so they are not altered by substitutions. The substitution $[b/a]$ can therefore only be affecting the truth of these conditions when $\alpha = a$ or when $a \in \text{fn}(A'_k) \setminus (Y \cup \{x\})$. Since we know a and b are assigned the same type by Γ and $\Gamma \subseteq \Gamma'$, condition (5) still holds when substituting a for b. Condition (6) holds by inductive hypotesis. The first disjunct of condition (7) depends only on types, which are not changed by the substitution, so it holds after applying it if and only if it holds before the application. To see that the second disjunct also holds after the substitution we observe that the *migratable* condition depends on x and $\text{fn}(A'_k) \cap Y$ which are preserved by the substitution; moreover, if $a \in \text{fn}(A'_k) \setminus \{\alpha\}$ then $\Gamma'(\text{fn}(A'_k) \setminus \{\alpha\}) = \Gamma'(\text{fn}(A'_k[b/a]) \setminus \{\alpha\})$.

This shows that the premises needed to derive $\Gamma', x:\tau'_x \vdash_{\mathcal{T}} \pi_{ij}.P_{ij}[b/a]$ are implied by our hypothesis, which completes the proof. □

Before we state the main theorem, we define the notion of P-safe type environment, which is a simple restriction on the types that can be assigned to names that are free at the top-level of a term.

Definition 6. A type environment Γ is said to be *P-safe* if for each $x \in \mathrm{fn}(P)$ and $(y:\tau) \in \mathrm{bn}_\nu(P)$, $\mathrm{base}(\Gamma(x)) < \mathrm{base}(\tau)$.

Theorem 1 (Subject Reduction)**.** *Let P and Q be two terms in $\mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ and Γ be a P-safe type environment. If $\Gamma \vdash_{\mathcal{T}} P$ and $P \to Q$, then $\Gamma \vdash_{\mathcal{T}} Q$.*

Proof. We will only prove the result for the case when $P \to Q$ is caused by a synchronising send and receive action since the $\boldsymbol{\tau}$ action case is similar and simpler. From $P \to Q$ we know that $P \equiv \nu W.(S \parallel R \parallel C) \in \mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ with $S \equiv (\overline{a}\langle b\rangle.\nu Y_s.S') + M_s$ and $R \equiv (a(x).\nu Y_r.R') + M_r$ the synchronising sender and receiver respectively; $Q \equiv \nu W Y_s Y_r.(S' \parallel R'[b/x] \parallel C)$. In what follows, let $W' = WY_sY_r$, $C = \prod_{h\in H} C_h$, $S' = \prod_{i\in I} S'_i$ and $R' = \prod_{j\in J} R'_j$, all normal forms.

For annotated terms, the type system is syntax directed: there can be only one proof derivation for each typable term. By Lemma 7.c, from the hypothesis $\Gamma \vdash_{\mathcal{T}} P$ we can deduce $\Gamma \vdash_{\mathcal{T}} \nu W.(S \parallel R \parallel C)$. The proof derivation for this typing judgment can only be of the following shape:

$$\frac{\Gamma W \vdash_{\mathcal{T}} S \quad \Gamma W \vdash_{\mathcal{T}} R \quad \forall h \in H.\, \Gamma W \vdash_{\mathcal{T}} C_h \quad \Psi}{\Gamma \vdash_{\mathcal{T}} \nu W.(S \parallel R \parallel C)} \tag{8}$$

where Ψ represents the rest of the conditions of the PAR rule.[3] The fact that P is typable implies that each of these premises must be provable. The derivation proving $\Gamma, W \vdash_{\mathcal{T}} S$ must be of the form

$$\frac{\dfrac{a:t_a[\tau_b] \in \Gamma W \quad b:\tau_b \in \Gamma W \quad \Gamma W \vdash_{\mathcal{T}} \nu Y_s.S'}{\Gamma W \vdash_{\mathcal{T}} \overline{a}\langle b\rangle.\nu Y_s.S'} \Psi_{M_s}}{\Gamma \vdash_{\mathcal{T}} \overline{a}\langle b\rangle.\nu Y_s.S' + M_s} \tag{9}$$

where $\Gamma W \vdash_{\mathcal{T}} \nu Y_s.S'$ is proved by an inference of the shape

$$\frac{\forall i \in I.\, \Gamma W Y_s \vdash_{\mathcal{T}} S'_i \qquad \forall i \in I.\, \Psi_{S'_i}}{\Gamma W \vdash_{\mathcal{T}} \nu Y_s.S'} \tag{10}$$

Analogously, $\Gamma W \vdash_{\mathcal{T}} R$ must be proved by an inference with the following shape

$$\frac{\dfrac{a:t_a[\tau_x] \in \Gamma W \quad \Gamma W, x:\tau_x \vdash_{\mathcal{T}} \nu Y_r.R' \quad \Psi_{R'}}{\Gamma W \vdash_{\mathcal{T}} a(x).\nu Y_r.R'} \Psi_{M_r}}{\Gamma W \vdash_{\mathcal{T}} a(x).\nu Y_r.R' + M_r} \tag{11}$$

and to prove $\Gamma W, x:\tau_x \vdash_{\mathcal{T}} \nu Y_r.R'$

$$\frac{\forall j \in J.\, \Gamma W, x:\tau_x, Y_r \vdash_{\mathcal{T}} R'_j \qquad \forall j \in J.\, \Psi_{R'_j}}{\Gamma W, x:\tau_x \vdash_{\mathcal{T}} \nu Y_r.R'} \tag{12}$$

We have to show that from this hypothesis we can infer that $\Gamma \vdash_{\mathcal{T}} Q$ or, equivalently (by Lemma 7.c), that $\Gamma \vdash_{\mathcal{T}} Q'$ where $Q' = \nu W Y_s Y_r.(S' \parallel R'[b/x] \parallel C)$. The derivation of this judgment can only end with an application of PAR:

$$\frac{\forall i \in I.\, \Gamma W' \vdash_{\mathcal{T}} S'_i \quad \forall j \in J.\, \Gamma W' \vdash_{\mathcal{T}} R'_j[b/x] \quad \forall h \in H.\, \Gamma W' \vdash_{\mathcal{T}} C_h \quad \Psi'}{\Gamma \vdash_{\mathcal{T}} \nu W'.(S' \parallel R'[b/x] \parallel C)}$$

In what follows we show how we can infer these premises are provable as a consequence of the provability of the premises of the proof of $\Gamma \vdash_{\mathcal{T}} \nu W.(S \parallel R \parallel C)$.

From Lemma 7.b and Name Uniqueness, $\Gamma W Y_s \vdash_{\mathcal{T}} S'_i$ from (10) implies $\Gamma W' \vdash_{\mathcal{T}} S'_i$ for each $i \in I$.

Let $\Gamma_r = \Gamma W, x:\tau_x$. We observe that by (9) and (11), $\tau_x = \tau_b$. From (11) we know that $\Gamma_r Y_r \vdash_{\mathcal{T}} R'_j$ which, by Lemma 8, implies $\Gamma_r Y_r \vdash_{\mathcal{T}} R'_j[b/x]$. By Lemma 7.b we can infer $\Gamma_r Y_r Y_s \vdash_{\mathcal{T}} R'_j[b/x]$ and by applying the same lemma again using $\mathrm{fn}(R'_j[b/x]) \subseteq \mathrm{dom}(\Gamma W Y_r Y_s)$ and Name Uniqueness we obtain $\Gamma W' \vdash_{\mathcal{T}} R'_j[b/x]$.

Again applying Lemma 7.b and Name Uniqueness, we have that $\Gamma W \vdash_{\mathcal{T}} C_h$ implies $\Gamma W' \vdash_{\mathcal{T}} C_h$ for each $h \in H$.

To complete the proof we only need to prove that for each $A \in \{S'_i \mid i \in I\} \cup \{R'_j \mid j \in J\} \cup \{C_h \mid h \in H\}$, $\Psi' = \forall (x:\tau_x) \in W'.\ x$ tied to A in $Q' \implies \mathrm{base}(\Gamma(\mathrm{fn}(A))) < \mathrm{base}(\tau_x)$ holds. This is trivially true by the hypothesis that Γ is P-safe. □

Theorem 2 (Invariance of $\mathcal{T}$-shapedness)**.** *Let P and Q be terms in $\mathcal{P}^{\mathcal{T}}_{\mathsf{nf}}$ such that $P \to Q$ and Γ be a P-safe environment such that $\Gamma \vdash_{\mathcal{T}} P$. Then, if P is $\mathcal{T}$-shaped then Q is $\mathcal{T}$-shaped.*

Proof. We will consider the input output synchronisation case as the τ action one is similar and simpler. We will further assume that the sending action $\overline{a}\langle b\rangle$ is such that $\nu(a:\tau_a)$ and $\nu(b:\tau_b)$ are both active restrictions of P, i.e. $(a:\tau_a) \in W$, $(b:\tau_b) \in W$ with $P \equiv \nu W.(S \parallel R \parallel C)$. The case when any of these two names is a free name of P can be easily handled with the aid of the assumption that Γ is P-safe.

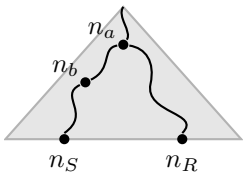
As in the proof of Theorem 1, the derivation of $\Gamma \vdash_{\mathcal{T}} P$ must follow the shape of (8).

From $\mathcal{T}$-shapedness of P we can conclude that both $\nu Y_s.S'$ and $\nu Y_r.R'$ are $\mathcal{T}$-shaped. We note that substitutions do not affect $\mathcal{T}$-compatibility since they do not alter the set of bound names and their type annotations. Therefore, we can infer that $\nu Y_r.R'[b/a]$ is $\mathcal{T}$-shaped. By Lemma 5 we know that $\varphi = \Phi(\nu W.(S \parallel R \parallel C)) \in \mathcal{F}[\![P]\!]$, $\varphi_r = \Phi(\nu Y_r.R'[b/a]) \in \mathcal{F}[\![\nu Y_r.R'[b/x]]\!]$ and $\varphi_s = \Phi(\nu Y_s.S') \in \mathcal{F}[\![\nu Y_s.S']\!]$. Let $\varphi_r = \varphi_{\mathrm{mig}} \uplus \varphi_{\neg\mathrm{mig}}$ where only φ_{mig} contains a leaf labelled with a term with b as a free name. These leaves will correspond to the continuations R'_j that migrate in $a(x).\nu Y_r.R'$, after the application of the substitution $[b/x]$. By assumption, inside P both S and R are in the scope of the restriction bounding a and S must also be in the scope of the restriction bounding b. Let $t_a = \mathrm{base}(\tau_a)$ and $t_b = \mathrm{base}(\tau_b)$, φ will contain two leaves n_S and n_R labelled with S and R respectively, having a common ancestor n_a labelled with (a, t_a); n_S will have an ancestor n_b labelled with (b, t_b). Let p_a, p_S and p_R be the paths in φ leading from a root to n_a, n_S and n_R respectively.

[3] Note that Ψ is trivially true by P-safety of Γ.

By $\mathcal{T}$-compatibility of φ, we are left with only two possible cases: either 1) $t_a < t_b$ or 2) $t_b < t_a$.

Let us consider case 1) first. The tree in φ to which the nodes n_S and n_R belong, would have the following shape:



Now, we want to transform φ, by manipulating this tree, into a forest φ' that is $\mathcal{T}$-compatible by construction and such that there exists a term $Q' \equiv Q$ with $\mathrm{forest}(Q') = \varphi'$, so that we can conclude Q is $\mathcal{T}$-shaped.

To do so, we introduce the following function, taking a labelled forest φ, a path p in φ and a labelled forest ρ and returning a labelled forest:

$$\mathrm{ins}(\varphi, p, \rho) := (N_\varphi \uplus N_\rho, \prec_\varphi \uplus \prec_\rho \uplus \prec_{\mathrm{ins}}, \ell_\varphi \uplus \ell_\rho)$$

where $n \prec_{\mathrm{ins}} n'$ if $n' \in \min_{\prec_\rho}(N_\rho)$, $\ell_\rho(n') = (y, t_y)$ and $n \in \max_{\prec_\varphi} \{m \in p \mid \ell_\varphi(m) = (x, t_x), t_x < t_y\}$. Note that for each n', since p is a path, there can be at most one n such that $n \prec_{\mathrm{ins}} n'$.

To obtain the desired φ', we first need to remove the leaves n_S and n_R from φ, as they represent the sequential processes which reacted, obtaining a forest φ_C. We argue that the φ' we need is indeed

$$\begin{aligned}\varphi' &= \mathrm{ins}(\varphi_1, p_S, \varphi_{\mathrm{mig}})\\ \varphi_1 &= \mathrm{ins}(\varphi_2, p_R, \varphi_{\neg\mathrm{mig}})\\ \varphi_2 &= \mathrm{ins}(\varphi_C, p_S, \varphi_s)\end{aligned}$$

It is easy to see that, by definition of ins, φ' is $\mathcal{T}$-compatible: φ_C, φ_s, $\varphi_{\neg\mathrm{mig}}$ and φ_{mig} are $\mathcal{T}$-compatible by hypothesis, ins adds parent-edges only when they do not break $\mathcal{T}$-compatibility.

To prove the claim we need to show that φ' is the forest of a term congruent to $\nu W Y_s Y_r.(S' \parallel R'[b/x] \parallel C)$. Let $R' = \prod_{j\in J} R'_j$, $J_{\mathrm{mig}} = \{j \in J \mid x \lhd_{\nu Y_r.R'} j\}$, $J_{\neg\mathrm{mig}} = J \setminus J_{\mathrm{mig}}$ and $Y'_r = \{(x:\tau) \in Y_r \mid x \in \mathrm{fn}(R'_j), j \in J_{\neg\mathrm{mig}}\}$. We know that no R'_j with $j \in J_{\neg\mathrm{mig}}$ can contain x as a free name so $R'_j[b/x] = R'_j$. Now suppose we are able to prove that conditions 1, 2 and 3 of Lemma 1 hold for φ_C, φ_1, φ_2 and φ'. Then we could use Lemma 1 to prove

a) $\varphi_C = \mathrm{forest}(Q_C)$, $Q_C \equiv Q_{\varphi_C} = \nu W.C$,
b) $\varphi_2 = \mathrm{forest}(Q_2)$, $Q_2 \equiv Q_{\varphi_2} = \nu W Y_s.(S' \parallel C)$,
c) $\varphi_1 = \mathrm{forest}(Q_1)$, $Q_1 \equiv Q_{\varphi_1} = \nu W Y_s Y'_r.(S' \parallel \prod_{j\in J_{\neg\mathrm{mig}}} R'_j \parallel C)$,
d) $\varphi' = \mathrm{forest}(Q')$, $Q' \equiv Q_{\varphi'} = \nu W Y_s Y_r.(S' \parallel R'[b/x] \parallel C) \equiv Q$

(it is straightforward to check that $\varphi_C, \varphi_2, \varphi_1$ and φ' have the right sets of nodes and labels to give rise to the right terms). We then proceed to check for each of the forests above that they satisfy conditions 1, 2 and 3, thus proving the theorem.

Condition 1 requires that only leafs are labelled with sequential processes, condition that is easily satisfied by all of the above forests since none of the operations involved in their definition alters this property and the forests φ, φ_s and φ_r satisfy it by construction.

Similarly, since $\nu W.(S \parallel R \parallel C)$ is a normal form it satisfies Name Uniqueness, 2 is satisfied as we never use the same name more than once.

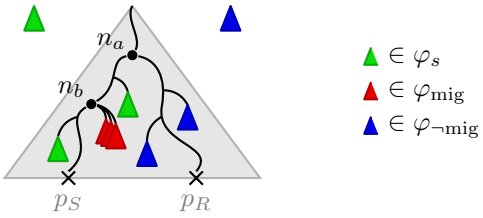
Condition 3 holds on φ and hence it holds on φ_C since the latter contains all the nodes of φ labelled with names.

Now consider φ_s: in the proof of Theorem 1 we established that $\Gamma \vdash_{\mathcal{T}} P$ implies that the premises $\Psi_{S'_i}$ from (10) hold, that is $\mathrm{base}(\Gamma W(\mathrm{fn}(S'_i))) < \mathrm{base}(\tau_x)$ holds for all S'_i for $i \in I$ and all $(x:\tau_x) \in Y_s$ such that $x \lhd_{\nu Y_s.S'} i$. Since $\mathrm{fn}(S'_i) \cap W \subseteq \mathrm{fn}(S')$ we know that every name $(w:\tau_w) \in W$ such that $w \in \mathrm{fn}(S'_i)$ will appear as a label $(w, \mathrm{base}(\tau_w))$ of a node n_w in p_S. Therefore, by definition of ins, we have that for each $n \in N_{\varphi_C}$, $n_w <_{\varphi_2} n$; in other words, in φ_2, every leaf in N_{φ_s} labelled with S'_i is a descendent of a node labelled with $(w, \mathrm{base}(\tau_w))$ for each $(w:\tau_w) \in W$ with $w \in \mathrm{fn}(S'_i)$. This verifies condition 3 on φ_2.

Similarly, by (12) the following premise must hold: $\mathrm{base}(\Gamma W(\mathrm{fn}(R'_j))) < \mathrm{base}(\tau_x)$ for all R'_j for $j \in J$ and all $(y:\tau_y) \in Y_r$ such that $y \lhd_{\nu Y_r.R'} j$. We can then apply the same argument we applied to φ_2 to show that condition 3 holds on φ_1.

From (11) and the assumption $t_a < t_b$, we can conclude that the following premise must hold: $\mathrm{base}(\Gamma W(\mathrm{fn}(R'_j) \setminus \{a\})) < t_a$ for each $j \in J$ such that R'_j is migratable in $a(x).\nu Y_r.R'$, i.e $j \in J_{\mathrm{mig}}$. From this we can conclude that for every name $(w:\tau_w) \in W$ such that $w \in \mathrm{fn}(R'_j[b/x])$ with $j \in J_{\mathrm{mig}}$ there must be a node in p_a (and hence in p_S) labelled with $(w, \mathrm{base}(\tau_w))$. Now, some of the leaves in φ_{mig} will be labelled with terms having b as a free name; we show that in fact every node in φ_{mig} labelled with a (y, t_y) is indeed such that $t_y < t_b$. From the proof of Theorem 1 and Lemma 8 we know that from the hypothesis we can infer that $\Gamma W \vdash_{\mathcal{T}} \nu Y_r.R'[b/x]$ and hence that for each $j \in J_{\mathrm{mig}}$ and each $(y:\tau_y) \in Y_r$, if y is tied to $R'_j[b/x]$ in $\nu Y_r.R'[b/x]$ then $\mathrm{base}(\Gamma W(R'_j[b/x])) < \mathrm{base}(\tau_y)$. By Lemma 6 we know that every root of φ_{mig} is labelled with a name (y, t_y) which is tied to each of the leaves in its tree. Therefore each such t_y satisfies $\mathrm{base}(\Gamma W(R'_j[b/x])) < t_y$. By construction, there exists at least one $j \in J_{\mathrm{mig}}$ such that $x \in \mathrm{fn}(R'_j)$ and consequently such that $b \in \mathrm{fn}(R'_j[b/x])$. From this and $b \in W$ we can conclude $t_b < t_y$ for t_y labelling a root in φ_{mig}. We can then conclude that $\{n_b\} = \max_{\prec_{\varphi_2}} \{m \in p_S \mid \ell_\varphi(m) = (z, t_z), t_z < t_y\}$ for each t_y labelling a root of φ_{mig}, which means that each tree of φ_{mig} is placed as a subtree of n_b in φ'. This verifies condition 3 for φ' completing the proof.

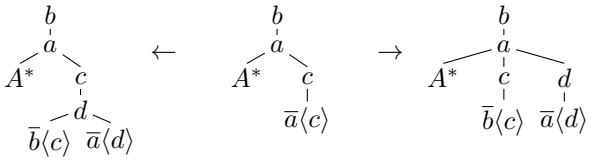
Pictorially, the tree containing n_S and n_R in φ is now transformed in the following tree in φ':



Case 2) — where $t_b < t_a$ — is simpler as the migrating continuations can be treated just as the non-migrating ones. □

To illustrate the role of φ_{mig}, $\varphi_{\neg\text{mig}}$ and the ins operation in the above proof, we show an example that would not be typable if we choose a simpler "migration" transformation.

Example 10. Consider the normal form $P = \nu a\, b\, c.(A^* \parallel \overline{a}\langle c\rangle)$ where $A = a(x).\nu d.(\overline{a}\langle d\rangle \parallel \overline{b}\langle x\rangle)$. To make types consistent we need annotations satisfying $a : t_a[t]$, $b : t_b[t]$, $c : t$ and $d : t$. Any $\mathcal{T}$ satisfying the constraints $t_b < t_a < t$ would allow us to prove $\emptyset \vdash_{\mathcal{T}} P$; let then $\mathcal{T}$ be the forest with $\mathsf{b} \prec \mathsf{a} \prec \mathsf{t}$ with $t_a = \mathsf{a}$, $t_b = \mathsf{b}$ and $t = \mathsf{t}$. Let $P' = \nu a\, b\, c\, d.(A^* \parallel \overline{a}\langle d\rangle \parallel \overline{b}\langle c\rangle)$ be the (only) successor of P. The following picture shows $\Phi(P)$ in the middle, on the left a forest in $\mathcal{F}[\![P']\!]$ extracted by just putting the continuation of A under the message, on the right the forest obtained by using ins on the non-migrating continuations of A:



Clearly, the tree on the left is not $\mathcal{T}$-compatible since c and d have the same base type t. Instead, the tree on the right can be obtained because ins inserts the non-migrating continuation as close to the root as possible.

Definition 7 (Typably Hierarchical term)**.** A normal form P is *typably hierarchical* if P is $\mathcal{T}$-shaped and $\Gamma \vdash_{\mathcal{T}} P$ for some finite forest $\mathcal{T}$ and P-safe environment Γ. A general π-term P is typably hierarchical if its normal form $\text{nf}(P)$ is.

Theorem 3 (Depth-boundedness)**.** *Every typably hierarchical term is depth-bounded.*

Proof. By Theorem 1 and Theorem 2 every term reachable from a typably hierarchical term P is $\mathcal{T}$-shaped. Then by Proposition 1 P is depth-bounded. □

B. Type inference

In this section we will show that it is possible to take any non-annotated normal form P and derive a forest $\mathcal{T}$ and an annotated normal form for P that can be typed under $\mathcal{T}$.

It is straightforward to see that inference is decidable: if a forest of base types can be found so that the typing derivation for P is successful, there exists a $\mathcal{T}$ with at most $|\text{bn}(P)|$ nodes and a P-safe environment Γ with $\text{dom}(\Gamma) = \text{fn}(P)$, such that $\Gamma \vdash_{\mathcal{T}} P$ and P is $\mathcal{T}$-shaped. Therefore, a naïve algorithm could enumerate all such forests—there are finitely many—and type check P against each. However a better algorithm is possible.

We start by annotating the term with type variables: each name x gets typed with a type variable $\mathfrak{t}_x$. Then we start the type derivation, collecting all the constraints on types along the way. If we can find a $\mathcal{T}$ and type expressions to associate to each type variable, so that these constraints are satisfied, the process can be typed under $\mathcal{T}$.

The constraints have two forms:

1) $\mathfrak{t}_x = t_x[\mathfrak{t}_y]$ where t_x is a base type variable;
2) $\text{base}(\mathfrak{t}_x) < \text{base}(\mathfrak{t}_y)$ which correspond to constraints over the corresponding base type variables, i.e. $t_x < t_y$.

Note that the P-safe condition on Γ translates to constraints of the second kind. The first kind of constraints can be solved using unification. If no solution exists, the process cannot be typed. This is the case of processes that cannot be *simply typed* [17]. If unification is successful we get a set of equations where the unknowns are the base type variables. Any assignment of those variables to nodes in a suitable forest that satisfies the constraints of the second kind would be a witness of typability.

We have at most n base type variables where n is the number of names occurring in P. There are at most $\frac{n(n-1)}{2}$ distinct independent constraints of the form $t_x < t_y$, which can be treated as uninterpreted propositions. By inspecting rules Par and In we observe that all the "tied-to" and "migratable" predicates do not depend on $\mathcal{T}$ so for any given P, the conjunction of constraints on base types generated in the proof derivation forms a 2-CNF formula with $O(n^2)$ boolean variables. Since 2-CNF satisfiability is linear in the number of variables [1], we obtain a $O(n^2)$ bound on satisfiability of the base type constraints. Once we prove satisfiability of these constraints, to prove P is typably hierarchical, it remains to prove that there exists a model $\mathcal{T}$ of the constraints so that P is $\mathcal{T}$-shaped. If a precise bound on the depth is needed, one can perform a search for the shallowest forest which is a model of the base type constraints such that P is $\mathcal{T}$-shaped. Otherwise, the search can be restricted to total orders.

V. Equivalence with NDCMA

After isolating a fragment of a process calculus, an interesting question is *can we find an automata based presentation of the same fragment?* In this section we give an answer to this question by relating the typably hierarchical fragment to a class of automata on data-words recently defined in [3]: *Nested Data Class Memory Automata* (NDCMA).

The original presentation of NDCMAs sees them as language recognition devices: they can recognise sets of data-words, that is sequences of symbols in $\Sigma \times \mathcal{D}$ where Σ is a finite alphabet and $\mathcal{D}$ is an infinite set of *data values*. Notably, (weak) NDCMAs are more expressive than Petri nets, while enjoying decidability of some verification problems. While *Class Memory Automata* [2] do not postulate any structure on $\mathcal{D}$, NDCMAs assume that it is equipped with an infinitely branching, finite height forest structure. We will make use of this forest structure to represent $\mathcal{T}$-compatible π-term forests.

We are primarily interested in establishing a tight relation between the transition systems of NDCMAs and typably hierarchical terms. Therefore we do not regard NDCMAs as language recognisation devices but simply as computational models. For this reason, our definition ignores the language-related components of the original definition of [3]: there is no finite alphabet Σ, no accepting control states, no accepting run. While in the language-theoretic formulation at each step in a run a letter and a data value must be read from the input string, here a transition can fire simply if *there exists* a data value satisfying the transition's precondition.

Definition 8 (NDCMA [3])**.** We define a *nested dataset* $(\mathcal{D}, \mathrm{pred}_{\mathcal{D}})$ to be a forest of infinitely many trees of level ℓ which is *full* in the sense that for each data value d of level less than ℓ, there are infinitely many data values d' whose parent is d.

A *class memory function* is a function $f\colon \mathcal{D} \to A \uplus \{\mathfrak{f}\}$ such that $f(d) = \mathfrak{f}$ for all but finitely many $d \in \mathcal{D}$; $\mathfrak{f}$ is a special symbol indicating a data value is fresh, i.e. has never been used before.

Fix a nested data set of level ℓ. A *Nested Data CMA of level* ℓ is a tuple $(\mathbb{Q}, \delta, q_0, f_0)$ where $\mathbb{Q}$ is a finite set of states, $q_0 \in \mathbb{Q}$ is the initial control state, $f_0\colon \mathcal{D} \to \mathbb{Q}_{\mathfrak{f}}$ is the initial class memory function satisfying $f_0(\mathrm{pred}(d)) = \mathfrak{f} \implies f_0(d) = \mathfrak{f}$, and δ is the transition relation. δ is given by a union $\delta = \bigcup_{i=1}^{\ell} \delta_i$ where each δ_i is a relation: $\delta_i \subseteq \mathbb{Q} \times (\mathbb{Q}_{\mathfrak{f}})^i \times \mathbb{Q} \times \mathbb{Q}^i$ and $\mathbb{Q}_{\mathfrak{f}}$ is defined as $\mathbb{Q} \cup \{\mathfrak{f}\}$. A configuration is a pair (q, f) where $q \in \mathbb{Q}$, and $f\colon \mathcal{D} \to \mathbb{Q}_{\mathfrak{f}}$ is a class memory function. The initial configuration is (q_0, f_0). The automaton can transition from configuration (q, f) to configuration (q', f'), written $(q, f) \to_{\mathcal{A}} (q', f')$, just if there exists a level-i data value d such that $(q, q_1, \ldots, q_i, q', q'_1, \ldots, q'_i) \in \delta$, for all $j \in \{1, \ldots, i\}$, $f(\mathrm{pred}^{i-j}(d)) = q_j$ and

$$f' = f\big[\,\mathrm{pred}^{i-1}(d) \mapsto q'_1,\ \ldots,\ \mathrm{pred}(d) \mapsto q'_{i-1},\ d \mapsto q'_i\,\big].$$

Given a nested dataset $\mathcal{D}$ we write $\mathrm{CMF}(\mathcal{D}, \mathbb{Q})$ for the set of all class memory functions from $\mathcal{D}$ to $\mathbb{Q}_{\mathfrak{f}}$.

We want to show that, in some strong sense, NDCMAs are equi-expressive to typably hierarchical π-terms. First we show an encoding from typeable π-terms, then we prove that a transition system generated from the NDCMA encoding is bisimilar to the transition system generated by the reduction semantics of the π-term. This result is quite strong in that it implies the equivalence of many decision problems of the two formalisms. It also offers a bridge between infinite-alphabet automata and decidable fragments of π-calculus.

A. Encoding Typably Hierarchical terms into NDCMA

We make a few simplifying assumptions on the term to be encoded as an NDCMA. First, we assume P is a closed normal form, i.e. $\mathrm{fn}(P) = \emptyset$, second we assume P contains no $\boldsymbol{\tau}$ action. It would be easy to support the general case but we only focus on the core case for conciseness. Fix a closed $\mathcal{T}$-shaped π-term P such that $\emptyset \vdash_{\mathcal{T}} P$, with $\ell = \mathrm{height}(\mathcal{T})$. We will construct a level-ℓ automaton $\mathcal{A}[\![P]\!]$ from P so that their transition systems are essentially bisimilar.

The intuition behind the encoding is as follows. A configuration (q, f) represents a π-term P by using f to label a finite portion of $\mathcal{D}$ so that it is isomorphic to a $\mathcal{T}$-compatible forest in $\mathcal{F}[\![P]\!]$. Our encoding proceeds in rounds. A single synchronisation step between two processes will be simulated by a predictable number of steps of the automaton. Since π-terms exhibit non-determinism, the automata in the image of the encoding need to be non-deterministic as well. We make use of the non-determinism of the automata model in a second way: in a reduction, the two synchronising processes are not in the same path in the syntax tree (they are both leaves by construction) but the automaton can only examine one path in $\mathcal{D}$ at a time; we then first guess the sender, mark the channel carrying its message, then select a receiver waiting on that channel (which will be in the path of both processes) and then spawn their continuations in the relevant places. This requires separate steps and could lead to spurious deadlocks when no process is listening over the selected channel. These deadlocked states can be pruned from the bisimulation by restricting the relevant transition system to those configurations where the control state is a distinguished state that signals that the intermediate steps of a synchronisation have been completed. A successful round follows very closely the operations used in the proof of Theorem 2.

A round starts from a configuration with control state q_{ready}, then goes trough a number of intermediate steps until it either deadlocks or reaches another configuration with control state q_{ready}. Only reachable configurations of $\mathcal{A}[\![P]\!]$ with q_{ready} as control state will correspond to reachable terms of P. Thus, given an automaton $\mathcal{A} = (\mathbb{Q}, \delta, q_{\text{ready}}, f_0)$, we define the transition relation $(\Rightarrow_{\text{ready}}) \subseteq \mathrm{CMF}(\mathcal{D}, \mathbb{Q})^2$ as the minimal relation such that $f \Rightarrow_{\text{ready}} f'$ if $(q_{\text{ready}}, f) \to_{\mathcal{A}} (q_1, f_1) \to_{\mathcal{A}} \cdots \to_{\mathcal{A}} (q_n, f_n) \to_{\mathcal{A}} (q_{\text{ready}}, f')$ where in the possibly empty sequence of (q_i, f_i), $q_i \neq q_{\text{ready}}$.

To encode a reachable term Q in a configuration (q_{ready}, f) we use f to represent the forest $\Phi(Q)$: roughly speaking we represent a node n of $\Phi(Q)$ labelled with l with a data value d mapped to a q_l by f. Since in general, due to the generation of unboundedly many names, there might be infinitely many such labels l we need to show that we can indeed use only a finite number of distinct labels to be able to represent them with control states. This is achieved by using the concept of derivatives. The set of *derivatives* of a term P is the set of sequential subterms of P, both active or not active. More formally, it is the set defined by the following function

$$\begin{aligned}
\mathrm{der}(\mathbf{0}) &:= \emptyset \\
\mathrm{der}(\boldsymbol{\nu} x.P) &:= \mathrm{der}(P) \\
\mathrm{der}(P \parallel Q) &:= \mathrm{der}(P) \cup \mathrm{der}(Q) \\
\mathrm{der}(M^*) &:= \{M^*\} \cup \mathrm{der}(M) \\
\mathrm{der}(M + M') &:= \{M + M'\} \cup \mathrm{der}(M) \cup \mathrm{der}(M') \\
\mathrm{der}(\pi.P) &:= \{\pi.P\} \cup \mathrm{der}(P)
\end{aligned}$$

Clearly, $\mathrm{der}(P)$ is a finite set. Every active sequential subterm of a term P' reachable from P is congruent to a $Q\sigma$ for some

substitution σ. When P is depth-bounded, we know from [8] that, there is a finite set of substitutions such that the substitution σ above can always be drawn from this set. The assumption that P is $\mathcal{T}$-shaped and typable allows us to be even more specific. Let $X_{\mathcal{T}} = \{\chi_t \mid t \in \mathcal{T}\}$ be a finite set of names, we define $\Delta_P := \{Q\sigma \mid Q \in \mathrm{der}(P), \sigma\colon \mathrm{fn}(Q) \to (X_{\mathcal{T}} \cup \mathrm{fn}(P))\}$.

Lemma 9. *Let P be a term such that $\mathrm{forest}(P)$ is $\mathcal{T}$-compatible. Then there exists a term Q such that $\mathrm{forest}(Q)$ is $\mathcal{T}$-compatible, Q is an α-renaming of P, $\mathrm{bn}_\nu(Q) \subseteq X_{\mathcal{T}}$ and each active sequential subterm of Q is in Δ_P.*

Proof. By definition of $\mathcal{T}$-compatible forest we have that in any path of $\mathrm{forest}(P)$ no two distinct nodes will have labels (x,t) (x',t) so α-renaming each restriction $(x:\tau)$ of P' to $(\chi_{\mathrm{base}(\tau)}:\tau)$ will yield the desired Q. □

Henceforth, we will write $\Phi'(P)$ for a relabelling of the forest $\Phi(P)$ such that its labels use only names in $X_{\mathcal{T}}$, as justified by Lemma 9.

Corollary 1. *If a term P is typably hierarchical, then every $P' \in \mathrm{Reach}(P)$ is congruent to a term Q such that $\mathrm{bn}_\nu(Q) \subseteq X_{\mathcal{T}}$ and each active sequential subterm of Q is in Δ_P.*

Proof. By Theorem 2 and Lemma 9. □

The transition relation of the automaton encoding of a term P is then derived from the set Δ_P.

Before we show how to construct the transitions of the automaton from the term, we define a relation $\sim$ between terms and class memory functions. This relation formalises how we encode the term as a labelling of data values, and will have a crucial role in proving the soundness of the encoding. Let Q be a term reachable from P and (q_{ready}, f) be a configuration of an automaton $\mathcal{A}$. Let $\varphi = \Phi'(Q)$, the relation $Q \sim f$ holds if and only if there exists an injective function $\iota\colon \mathrm{nodes}(\varphi) \to \mathcal{D}$ such that for all $n \in \mathrm{nodes}(\varphi)$:

i) if $\iota(n) = d$, $n' \prec_\varphi n$ and $\iota(n') = d'$ then $d' = \mathrm{pred}(d)$;
ii) if n is labelled with (χ_i, t) then $f(\iota(n)) = \chi_i$;
iii) if n is labelled with a sequential process Q' then $f(\iota(n)) = Q'$;
iv) for each d such that $f(d) \neq \mathfrak{f}$ either there is an n such that $\iota(n) = d$ or $f(d) = q_\dagger$.

Let us now describe how we can simulate reduction steps of a π-term with transitions in a NDCMA. In encoding a π-term's semantics into the transition relation of a NDCMA, we need to overcome the differences in the primitive steps allowed in the two models. Simulating a π-calculus synchronisation requires matching two paths, leading to the two reacting sequential terms, in $\mathcal{D}$ at the same time. A step in the automata semantics can only manipulate a single path, so we will need to split the detection of a redex in two phases: finding the sender, then finding a matching receiver. Moreover, finding a redex requires detecting that the path under consideration contains a node labelled with the synchronising channel and one with the appropriate sequential term, ignoring how many and which other nodes are in between them. To succinctly represent this operation, we introduce the following notation. Fix a set $\mathbb{Q}$ including $q, q', l_1, \ldots, l_n, l'_1, \ldots, l'_n, l$. We associate to the expression $[q, l_1 \ldots l_n] \to [q', l'_1 \ldots l'_n]$ the set of transitions

$$\begin{aligned}&\mathrm{tran}_{\mathbb{Q}}([q, l_1 \ldots l_n] \to [q', l'_1 \ldots l'_n]) := \\ &\big\{(q, q_1, \ldots, q_m, q', q'_1, \ldots, q'_m) \in \mathbb{Q}^{2m+2} \mid \exists\, i_1 \ldots i_m. \\ &\qquad 1 \le i_1 < \cdots < i_n \le m, q_{i_j} = l_j, q'_{i_j} = l'_j\big\}.\end{aligned}$$

When the sequence $l_1, \ldots, l_n$ is empty, the expression simply means that the automaton may go from a configuration (q, f) to (q', f) with no condition (nor effect) on f. Similarly, we associate to the expression $[q, l_1 \ldots l_n; \mathfrak{f}] \to [q', l'_1 \ldots l'_n; l]$ the set of transitions

$$\begin{aligned}&\mathrm{tran}_{\mathbb{Q}}([q, l_1 \ldots l_n; \mathfrak{f}] \to [q', l'_1 \ldots l'_n; l]) := \\ &\big\{(q, q_1, \ldots, q_m, \mathfrak{f}, q', q'_1, \ldots, q'_m, l) \in \mathbb{Q}^{2m+4} \mid \exists\, i_1 \ldots i_m. \\ &\qquad 1 \le i_1 < \cdots < i_n = m, q_{i_j} = l_j, q'_{i_j} = l'_j\big\}.\end{aligned}$$

Note that the sequence $l_1, \ldots, l_n$ may be empty, in which case the data value labelled with $\mathfrak{f}$ is selected among the level-1 ones. The set of states mentioned in an expression is $\mathrm{states}([q, l_1 \ldots l_n] \to [q', l'_1 \ldots l'_n]) := \{q, q', l_1, \ldots, l_n, l'_1, \ldots, l'_n\}$ and $\mathrm{states}([q, l_1 \ldots l_n; \mathfrak{f}] \to [q', l'_1 \ldots l'_n; l]) := \{q, q', l, l_1, \ldots, l_n, l'_1, \ldots, l'_n\}$.

To define the transitions of the encoding of a term, we make use of some auxiliary definitions generating sets of transition expressions.

$\textsc{Setup}(q, q', l, \varphi)$ adds to the path leading to a data value labelled with l, the nodes corresponding to a forest $\varphi \in \mathcal{F}[\![Q]\!]$ for some Q. These transitions are deterministic in the sense that a configuration (q, f) with only one data value labelled with l will transition through all the transitions dictated by $\textsc{Setup}(q, q', l, l', \varphi)$ reaching (q', f'). Formally, suppose, for some j and k, $\varphi = \{(x_1, \tau_1)[\varphi_1], \ldots, (x_j, \tau_j)[\varphi_j]\} \cup \{Q_1[], \ldots, Q_k[]\}$ where all x_i are in $X_{\mathcal{T}}$ and all $Q_i \in \Delta_P$. Then $\textsc{Setup}$ is defined as follows:

$$\begin{aligned}\textsc{Setup}(q, q', l, l', \varphi) := {} & \{[q, l; \mathfrak{f}] \to [q_1, l; Q_1^{\text{ready}}]\} \\ & \cup \{[q_i, l; \mathfrak{f}] \to [q_{i+1}, l; Q_{i+1}^{\text{ready}}] \mid 1 \le i \le j\} \\ & \cup \{[q_j, l; \mathfrak{f}] \to [q'_1, l; Q_j^{\text{ready}}]\} \\ & \cup \{[q'_i, l; \mathfrak{f}] \to [q''_i, l; x_i^{\text{set}}] \mid 1 \le i \le k\} \\ & \cup \bigcup_{i=1}^{k} \textsc{Setup}(q''_i, q'_{i+1}, x_i^{\text{set}}, x_i, \varphi_i) \\ & \cup \{[q'_{k+1}, l] \to [q', l']\}\end{aligned}$$

where for all $1 \le i \le j$ and all $1 \le i' \le k$, $q_i, q'_{i'}, q''_{i'}, q'_{k+1}$ are fresh intermediate control states. in the sense that they are only mentioned in the transitions generated by that specific application of $\textsc{Setup}$. We allow l to be the empty sequence, in which case l' needs to be the empty sequence as well.

Similarly, we define $\textsc{Spawn}(q, q', l, l', \varphi)$ to be the set of transitions needed to append each tree in φ to nodes in the path leading to a data value d labelled with l; the operation starts at control state q and ends at control state q' with the label for d updated to l'. Each tree is appended to the node with the

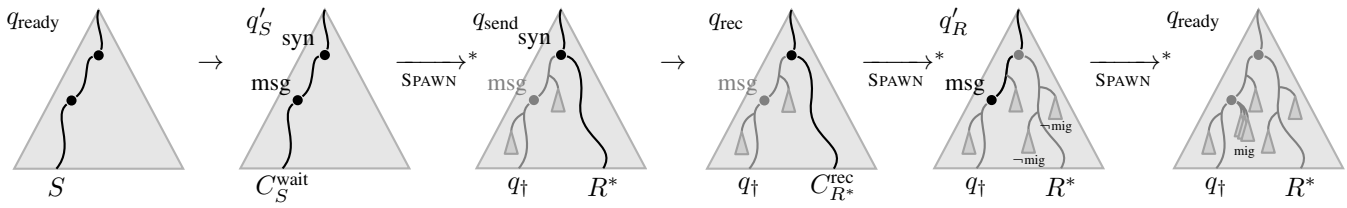


Figure 5. A schema of the transitions simulating a synchronisation in the automaton encoding of a term. The trees represent the class memory functions associated with configurations in the run of the automaton. The run simulates a sender synchronising with a replicated receiver. The two displayed nodes in the path leading to S are the ones labelled with the names of, from top to bottom, the synchronisation channel and the exchanged message.

lowest level such that every name mentioned in its leaves is an ancestor of such node. Since a single transition can add only one node of φ, we need a number of transitions to complete the operation; these transitions will however be deterministic in the same sense as the ones required to complete a SETUP operation. Formally, let the forest $\varphi = \Phi'(D)$ consist of trees $\theta_1, \ldots, \theta_k$, for a term $D \in \Delta_P$. We can precompute, for each θ_i, the base type $t_i := \min_{\prec_\mathcal{T}} \{t \mid \chi_t \in \text{fn}(A), n \in N_{\theta_i}, \ell_{\theta_i}(n) = A\}$ when defined. For each label $\chi_t \in X_\mathcal{T}$ we also have a label χ_t^{sp} we write $\chi(\theta_i)$ (resp. $\chi^{\text{sp}}(\theta_i)$) for χ_{t_i} (resp. $\chi_{t_i}^{\text{sp}}$) when t_i is defined, or the empty sequence when t_i is undefined (e.g. when θ_i does not have free variables). Then $\text{SPAWN}(q, q', l, l', \varphi)$ is the set of transition expressions defined as follows:

$$\begin{aligned}&\text{SPAWN}(q_0, q', l, l', \varphi) := \\ &\quad \{[q_{i-1}, \chi(\theta_i)\, l] \to [q'_{i-1}, \chi^{\text{sp}}(\theta_i)\, l] \mid 1 \le i \le k\} \\ &\quad \cup \bigcup_{i=1}^{k} \text{SETUP}(q'_{i-1}, q_i, \chi^{\text{sp}}(\theta_i), \chi(\theta_i), \theta_i)\end{aligned}$$

where for all $1 < h \le k$, q_h, q'_h are fresh.

We define for each $D \in \Delta_P$ the set of transition expressions $\text{REACT}(D)$ representing the steps needed to simulate in the automaton the potential reactions of D.

$$\begin{aligned}\text{REACT}(M) &:= \text{REACT}_{q_\dagger}^M(M) \\ \text{REACT}(M^*) &:= \text{REACT}_{M^*}^{M^*}(M)\end{aligned}$$

The set of transition expressions $\text{REACT}_q^D(M)$ collects all the potential reactions of M as a choice of D; the label q is the one that should be associated with the "consumed" term D after a reaction has been completed. The transitions simulating a replicated component will not mark, as the ones for non replicated terms, the reacted term with $q_\dagger$, which will represent "garbage" inert nodes in f. The term $\mathbf{0}$ cannot initiate any step and a choice may do any action that one of its choices can:

$$\begin{aligned}\text{REACT}_q^D(\mathbf{0}) &:= \emptyset \\ \text{REACT}_q^D(M + M') &:= \text{REACT}_q^D(M) \cup \text{REACT}_q^D(M')\end{aligned}$$

Any sender can initiate a synchronisation from the ready state:

$$\begin{aligned}&\text{REACT}_q^D(\overline{\chi_t}\langle\chi_{t'}\rangle.C) := \\ &\quad \{[q_{\text{ready}}, \chi_t\chi_{t'}D] \to [q', \chi_t^{\text{syn}}\chi_{t'}^{\text{msg}}C^{\text{wait}}] \mid t < t'\} \\ &\quad \cup \{[q_{\text{ready}}, \chi_{t'}\chi_t D] \to [q', \chi_{t'}^{\text{msg}}\chi_t^{\text{syn}}C^{\text{wait}}] \mid t > t'\} \\ &\quad \cup \text{SPAWN}(q', q_{\text{send}}, C^{\text{wait}}, q, \Phi'(C)).\end{aligned}$$

where q' is fresh. Here, the state q_{send} signals that we are in the middle of a synchronisation, where the sender is committed but a receiver has yet to be selected.

For the case of an input prefix $M = \chi_t(x).C$ we distinguish two cases: when the base type of χ_t is greater than the base type of x no migration occurs, otherwise part of the continuation needs to be spawned in the sender's path. In the case when the base type of χ_t is greater than the base type of x, we set

$$\begin{aligned}&\text{REACT}_q^D(\chi_t(x).C) := \\ &\quad \{[q_{\text{send}}, \chi_t^{\text{syn}}\chi_{t'}^{\text{msg}}D] \to [q_{\text{rec}}, \chi_t\chi_{t'}C^{\text{rec}}] \mid t < t' \in \mathcal{T}\} \\ &\quad \{[q_{\text{send}}, \chi_{t'}^{\text{msg}}\chi_t^{\text{syn}}D] \to [q_{\text{rec}}, \chi_{t'}\chi_t C^{\text{rec}}] \mid t > t' \in \mathcal{T}\} \\ &\quad \cup \text{SPAWN}(q_{\text{rec}}, q_{\text{ready}}, C^{\text{rec}}, q, \Phi'(C)).\end{aligned}$$

In the case when the base type of χ_t is greater than the base type of x, more transitions are required. First, we precompute for each $M = \chi_t(x).C$ as above and $t < t' \in \mathcal{T}$, the two forests $\varphi_{\text{mig}}(C, t')$ and $\varphi_{\neg\text{mig}}(C)$ such that $\Phi'(C[\chi_{t'}/x]) = \varphi_{\text{mig}}(C, t') \uplus \varphi_{\neg\text{mig}}(C)$ and $\varphi_{\text{mig}}(C, t')$ contains all the nodes labelled with sequential terms tied to $\chi_{t'}$ in $C[\chi_{t'}/x]$. As we have shown in the proof of Theorem 2, by virtue of Lemma 2, $\varphi_{\text{mig}}(C, t')$ and $\varphi_{\neg\text{mig}}(C)$ are indeed disjoint. Then we set:

$$\begin{aligned}&\text{REACT}_q^D(\chi_t(x).C) := \\ &\quad \{[q_{\text{send}}, \chi_t^{\text{syn}}D] \to [q_{\text{rec}}, \chi_t C^{\text{rec}}]\} \\ &\quad \cup \text{SPAWN}(q_{\text{rec}}, q', C^{\text{rec}}, q, \varphi_{\neg\text{mig}}(C)) \\ &\quad \cup \bigcup_{t' \in \mathcal{T}} \text{SETUP}(q', q_{\text{ready}}, \chi_{t'}^{\text{msg}}, \chi_{t'}, \varphi_{\text{mig}}(C, t'))\end{aligned}$$

where q' is a fresh intermediate control state. Figure 5 illustrates the steps the automaton performs when simulating a synchronisation.

Definition 9 (Automaton encoding)**.** The automaton encoding of a typably hierarchical term P is the NDCMA $\mathcal{A}[\![P]\!] = (\mathbb{Q}, \delta, q_{\text{ready}}, f)$ where $\text{TR} = \bigcup \{\text{REACT}(D) \mid D \in \Delta_P\}$, $\mathbb{Q} = \text{states}(\text{TR})$, $\delta = \text{tran}_\mathbb{Q}(\text{TR})$ and f is an arbitrary class memory function such that $P \sim f$.

B. Soundness of the encoding

In this section we will show that the transition system of the semantics of P is bisimilar to the one of $\mathcal{A}$ when restricting it to configurations with control state equal to q_{ready}.

A transition system is a tuple $(S, \to, s)$ where S is a set of configurations, $(\to) \subseteq (S \times S)$ is the transition relation and $s \in$

S is the initial state. Two transition systems $(S_1, \to_1, s_1)$ and $(S_2, \to_2, s_2)$ are said to be *bisimilar* if there exists a relation $(\approx) \subseteq S_1 \times S_2$ such that $s_1 \approx s_2$ and $\approx$ is a *bisimulation*, that is, if $s \approx t$ then: (A) for each $s' \in S_1$ such that $s \to_1 s'$ there is a $t' \in S_2$ such that $t \to_2 t'$ and $s' \approx t'$; (B) for each $t' \in S_2$ such that $t \to_2 t'$ there is a $s' \in S_1$ such that $s \to_1 s'$ and $s' \approx t'$. Establishing that two transition systems are bisimilar implies that a wide class of properties are preserved across bisimilar states. For our purposes, proving that the automaton encoding of a term gives rise to a bisimilar transition system has the important consequence that reachability can be reduced from one model to the other.

Theorem 4. *The transition system* $(\mathrm{CMF}(\mathcal{D}, \mathbb{Q}), \Rightarrow_{ready}, f_0)$ *induced by the automaton* $\mathcal{A}[\![P]\!] = (\mathbb{Q}, \delta, q_{ready}, f_0)$ *obtained from a closed typably hierarchical term* P*, is bisimilar to the transition system of the reduction semantics of* P*,* $(\mathrm{Reach}(P), \to, P)$.

The result is proved by showing that the relation $\sim$ defined above, is a bisimulation that relates the initial states of the two transition systems. By definition of $\mathcal{A}[\![P]\!]$ we have $P \sim f_0$. Showing that $\sim$ is indeed a bisimulation amounts to showing that if $Q \sim f$ then:

(A) for each Q' such that $Q \to Q'$ there is a f' such that $f \Rightarrow_{\text{ready}} f'$ and $Q' \sim f'$;
(B) for each f' such that $f \Rightarrow_{\text{ready}} f'$ there is a Q' such that $Q \to Q'$ and $Q' \sim f'$.

To show this holds we rely on the hypothesis that $Q \sim f$ to get a ι relating $\Phi'(Q)$ and f. The proof then closely follows the constructions in the proof of Theorem 2. If $Q \to Q'$ we can find two nodes n_S and n_R in $\Phi'(Q)$ labelled with the sender and receiver processes responsible for the reduction; they will share an ancestor n_a labelled (χ_t, t) corresponding to the channel on which they are synchronising. On the automaton side, we have that (q_{ready}, f) matches the rule generated from the sender by selecting the data value $d_S = \iota(n_S)$, a data value d_b corresponding to the name being sent and $d_a = \iota(n_a)$. This leads to (q', f) where $f'(d_a) = \chi_t^{\text{syn}}$, $f'(d_b) = \chi_{t'}^{\text{msg}}$, $f'(d_S) = S'^{\text{wait}}$. From here only one of the transitions generated from SPAWN of the continuation is enabled as there is only one node marked with 'wait'. The transitions are deterministic from here until a configuration (q_{send}, f') is reached with f' representing the initial forest with the continuation of the sender added and with the node of the sender updated with either $q_\dagger$ or the sender itself if it is a replicated component. At this point there is only one data value marked with 'syn' and the only transitions from q_{send} are the ones generated from a process that can receive from the marked channel. We can pick the rule that has been generated from the receiver involved in the reduction from Q to Q' and go to a configuration with control state q_{rec}. From this configuration the transitions are deterministic. The next configuration reached with control state q_{ready} is bisimilar to Q' by tracing the effects these transitions have on the class memory function. Fresh data values get assigned labels compatible with the non migrating continuations of the receiver first, and then the migrating ones as children of d_b; data values with meaningless labels get assigned the label $q_\dagger$.

To prove (B) we proceed similarly. Every reduction sequence from (q_{ready}, f) to (q_{ready}, f') must start with a transition to a configuration with control state q_{send}, which is generated by rules extracted from a sender S labelling a data value d_S; since $Q \sim f$ we know that $n_S = \iota^{-1}(d_S)$ is labelled with S in $\Phi'(Q)$, hence S is an active sequential process of Q. To complete this part of the proof we only need to follow the transitions of the automaton in the same way as done for the previous point, and note that the only way the automaton can reach a configuration with control state q_{ready} from (q_{ready}, f) is by selecting a receiver that can synchronise with the selected sender. This is important because there may be transitions from (q_{ready}, f) corresponding to selecting a sender trying to synchronise on a channel on which no receiver is listening. This transition would lead to a deadlocked configuration (one with no successors) but never going through a configuration with control state q_{ready}.

C. Encoding of NDCMA into Typably Hierarchical terms

In this section we sketch how an NDCMA can be encoded into a bisimilar typably hierarchical π-term.

Similarly as the encoding in the opposite direction, the π-calculus encoding of an automaton $\mathcal{A}$ will represent a reachable configuration (q, f) using the forest of a reachable term P. A term representing a reachable configuration may need to execute several steps before reaching another term representing a successor configuration.

Fix an automaton $(\mathbb{Q}, \delta, q_0, f_0)$. For simplicity we show the case where $\forall d.\ f_0(d) = \mathfrak{f}$, the general case follows the same scheme. First we note that every transition in δ_i is of the form

$$(q_0, q_1 \dots q_j, \underbrace{\mathfrak{f}, \dots, \mathfrak{f}}_{i-j}, q'_0, q'_1 \dots q'_i)$$

for some $1 \le j \le i$, where $q_k \in \mathbb{Q}$ for all $0 \le k \le j$. Instead of using the partition $\delta = \bigcup_{i=1}^{\ell} \delta_i$ we re-partition the transition relation as $\delta = \bigcup_{j=0}^{\ell} \theta_j$ where

$$\theta_j := \bigcup_{i=j}^{\ell} \{(q_0, q_1 \dots q_j, \underbrace{\mathfrak{f}, \dots, \mathfrak{f}}_{i-j}, q'_0, q'_1 \dots q'_i) \in \delta_i\}$$

(fixing $\delta_0 = \emptyset$ for uniformity). We introduce a channel name c_q^i for each $q \in \mathbb{Q}$ and each level of the automaton i. Our encoding will show no mobility, so each such channel c will have type t_c, hence no message will be exchanged on synchronisation; we abbreviate this kind of synchronisation with $c.P$ and $\overline{c}.Q$.[4] Let $\mathcal{C}^i := \{(c_q^i : t_{c_q^i}) \mid q \in \mathbb{Q}\}$. Given a transition $tr \in \theta_j$ where $tr = (q_0, q_1 \dots q_j, \mathfrak{f}, \dots, \mathfrak{f}, q'_0, q'_1 \dots q'_i)$ we define the

[4]It is easy to see that this can be accommodated in our syntax by assuming a global name r, typed with a type t_r that is set to be the parent of each root in $\mathcal{T}$; a synchronisation over a channel $c : t_c[t_r]$ without exchanging a message is then represented by $c(x).P$ and $\overline{c}\langle r \rangle.Q$ with $x \notin \mathrm{fn}(P)$.

term A_{tr} to be

$$A_{tr} := c^0_{q_0}.\cdots.c^j_{q_j}.\nu\mathcal{C}^{j+1}.\cdots\nu\mathcal{C}^i.\left(\prod_{k=0}^{i}\overline{c}^k_{q'_k} \parallel \prod_{k=j+1}^{i} P_{\theta_k}\right)$$

where $P_{\theta_j} := \prod_{tr\in\theta_j}(A_{tr})^*$ and $\prod_{k=i+1}^{i} P_{\theta_k} = \mathbf{0}$. Note that these definitions are well-defined since they are not recursive. The *π-term encoding* of the NDCMA $\mathcal{A} = (\mathbb{Q}, \delta, q_0, f_0)$ is then defined as $\mathcal{P}[\![\mathcal{A}]\!] := \nu\mathcal{C}^0.(P_{\theta_0} \parallel \overline{c}^0_{q_0})$.

Similarly to our previous result, the encoding needs more than one step to simulate a single transition of the automaton. Hence, to state the result on the correspondence between the semantics of the automaton and its encoding, we define a derived transition system on π-terms as follows. Let P and Q be two π-terms such that $P \to^+ Q$, if $P \equiv \nu\mathcal{C}^0.(\overline{c} \parallel P')$ and $Q \equiv \nu\mathcal{C}^0.(\overline{c}' \parallel Q')$ with c, $c' \in \mathcal{C}^0$, and none of the intermediate processes in the reduction from P to Q is in that form, then $P \Rightarrow_{\mathcal{C}^0} Q$. Note that even after α-renaming a term in the encoding, we would be able to pinpoint names from each $\mathcal{C}^i$ by looking at their types, as α-renaming does not affect type annotations.

Theorem 5. *The transition system generated by the semantics of a level-ℓ NDCMA $\mathcal{A}$ and the transition system $\Rightarrow_{\mathcal{C}^0}$ with $\mathcal{P}[\![\mathcal{A}]\!]$ as initial state, are bisimilar.*

Proof. Fix an NDCMA $\mathcal{A} = (\mathbb{Q}, \delta, q_0, f_0)$ with $\delta = \bigcup_{0\le j\le \ell}\theta_j$ as before. We prove the theorem by exhibiting a bisimulation relation $(\backsim) \subseteq (\mathbb{Q} \times (\mathcal{D} \rightharpoonup \mathbb{Q}_\mathfrak{f})) \times \mathrm{Reach}(\mathcal{P}[\![\mathcal{A}]\!])$ between the two transition systems. For a class memory function $f\colon \mathcal{D} \to \mathbb{Q}_\mathfrak{f}$, let $f(\mathcal{D})$ be the $\mathbb{Q}$-labelled forest with the set $N = \{d \in \mathcal{D} \mid f(d) \neq \mathfrak{f}\}$ as nodes, each labelled with $f(d)$ and with $\mathrm{pred}_\mathcal{D}$ restricted to N as parent relation. We first define a hierarchy of relations $\backsim_i$ between $\mathbb{Q}$-labelled forests and π-terms, for $0 \le i \le \ell$, as follows: $q[\{\varphi_1, \ldots, \varphi_n\}] \backsim_i \nu\mathcal{C}^i.(P_{\theta_i} \parallel \overline{c}^i_q \parallel \prod_{1\le j\le n} P_j)$ if, for all $1 \le j \le n$, $\varphi_j \backsim_{i+1} P_j$. Since n must be 0 for $i = \ell$, the relation is well-defined. Let $P \in \mathrm{Reach}(\mathcal{P}[\![\mathcal{A}]\!])$ and (q, f) be a reachable configuration of $\mathcal{A}$. Then $(q, f) \backsim P$ if there exists a $P' \equiv P$ such that $q_0[f(\mathcal{D})] \backsim_0 P'$. To show that $\backsim$ is indeed a bisimulation, we have to prove that if $(q, f) \backsim P$ then:

(A) for each (q', f') such that $(q, f) \to_\mathcal{A} (q', f')$ there is a P' such that $P \Rightarrow_{\mathcal{C}^0} P'$ and $(q', f') \backsim P'$;
(B) for each P' such that $P \Rightarrow_{\mathcal{C}^0} P'$ there is a (q', f') such that $(q, f) \to_\mathcal{A} (q', f')$ and $P' \backsim (q', f')$.

To prove (A) we proceed as follows; suppose $(q, f) \to_\mathcal{A} (q', f')$ is an application of a transition $t = (q, q_1 \ldots q_j, \mathfrak{f}, \ldots, \mathfrak{f}, q', q'_1 \ldots q'_i) \in \theta_j$ then the forest $q[f(\mathcal{D})]$ has a path from the root to a leaf labelled with $q, q_1, \ldots, q_j$, which, by definition of $\backsim$, implies that P is congruent to a term with the following shape:

$$\nu\mathcal{C}^0.(R_0 \parallel \overline{c}^0_q \parallel \nu\mathcal{C}^1.(R_1 \parallel \overline{c}^1_{q_1} \parallel \cdots \nu\mathcal{C}^j.(R_j \parallel \overline{c}^j_{q_j} \parallel P_{\theta_j})\cdots).$$

By construction, $P_{\theta_j} \equiv (A_{tr})^* \parallel R$ and A_{tr} is a process inputting once from c^0_q then once from each $c^k_{q_k}$ in sequence. From the shape of P we can conclude all of these input prefixes can synchronise with the dual $\overline{c}^k_{q_k}$ processes in parallel with them, activating, in $j+1$ steps, the continuation $C = \nu\mathcal{C}^{j+1}.\cdots\nu\mathcal{C}^i.\left(\overline{c}^0_{q'} \parallel \prod_{k=1}^{\ell}\overline{c}^k_{q'_k} \parallel P_{\theta_{j+1}}\right)$, yielding the process

$$P' \equiv \nu\mathcal{C}^0.(R_0 \parallel \overline{c}^0_{q'} \parallel \nu\mathcal{C}^1.(R_1 \parallel \overline{c}^1_{q'_1} \parallel \cdots \nu\mathcal{C}^i.(R_i \parallel \overline{c}^j_{q'_j})\cdots)$$

where for k between $j+1$ and i, $R_k = P_{\theta_k}$. Now consider the forest $q'[f'(\mathcal{D})]$: it coincides with $q[f(\mathcal{D})]$ except on the path we singled out, now labelled with $q', q'_0, \ldots, q'_j$ and continuing to a leaf with nodes labelled $q'_{j+1}, \ldots, q'_i$. It is easy to see that $q'[f'(\mathcal{D})] \backsim P'$.

To prove (B) one can proceed similarly, by observing that even if $\mathcal{P}[\![\mathcal{A}]\!]$ can perform some reductions which deadlock that do not correspond to reductions of the automaton, these steps cannot lead to a state with $\overline{c}^0_{q'}$ as one of the active sequential processes. This claim is supported by the following easy to verify invariant: in any term P reachable from $\mathcal{P}[\![\mathcal{A}]\!]$, for each bound name c in P there is at most one active sequential subterm of P outputting on c. This is satisfied by $\mathcal{P}[\![\mathcal{A}]\!]$ and preserved by reduction. □

Theorem 6. *$\mathcal{P}[\![\mathcal{A}]\!]$ is typably hierarchical.*

Proof. Assume an arbitrary strict total order $<_\mathbb{Q}$ on the automaton's control states; let then $(\mathcal{T}, \prec)$ be the forest with nodes $\mathcal{T} = \{t_{c^i_q} \mid 0 \le i \le \ell, q \in \mathbb{Q}\}$ and $t_{c^i_q} \prec t_{c^i_{q'}}$ if $q <_\mathbb{Q} q'$, and $t_{c^i_q} \prec t_{c^{i+1}_{q'}}$ if q and q' are respectively the maximum and minimum states with respect to $<_\mathbb{Q}$. It can be proved that $\emptyset \vdash_\mathcal{T} \mathrm{nf}(\mathcal{P}[\![\mathcal{A}]\!])$: since no messages are exchanged over channels, the constraints on types are trivially satisfied; for the same reason, no sequential term under an input prefix is migratable, making all the base type constraints in rule In trivially valid. The base type inequalities of rule Par are also satisfied since in A_{tr} for $tr \in \theta_j$, every P_{θ_k} might be tied to any channel c in $\mathcal{C}^{j+1} \cup \ldots \cup \mathcal{C}^i$ but can only have as free names channels in $\mathcal{C}^h$ with $h \le j$, which all have base types smaller than c. □

VI. Related Work

Depth boundedness in the π-calculus was first proposed in [9] and later studied in [8] where it is proved that depth-bounded systems are well-structured transition systems. In [20] it is further proved that (forward) coverability is decidable even when the depth bound k is not known *a priori*. In [21] an approximate algorithm for computing the *cover set*—an over-approximation of the set of reachable terms—of a system of depth bounded by k is presented. All these analyses rely on the assumption of depth-boundedness and may even require a known bound on the depth to terminate.

Several other interesting fragments of the π-calculus have been proposed in the literature, such as name bounded [6], mixed bounded [10], and structurally stationary [9]. Typically defined by a non-trivial condition on the set of reachable terms – a *semantic* property, membership becomes undecidable. Links with Petri nets via encodings of proper subsets of depth-bounded systems have been explored in [10]. Our type system can prove depth-boundedness for processes that are breadth

and name unbounded, and which cannot be simulated by Petri nets. Recently Hüchting et al. [18] proved several relative classification results between fragments of π-calculus. Using Karp-Miller trees, they presented an algorithm to decide if an arbitrary π-term is bounded in depth by a given k. The construction is based on an (accelerated) exploration of the state space of the π-term which can be computationally expensive. By contrast, our type system uses a very different technique leading to a quicker algorithm, at the expense of precision. Our forest-structured types can also act as specifications, offering more intensional information to the user than just a bound k.

Our type system is based on Milner's sorts for the π-calculus [12], later refined into I/O types [16] and their variants [17]. Based on these types is a system for termination of π-terms [5] that uses a notion of levels, enabling the definition of a lexicographical ordering. Our type system can also be used to determine termination of π-terms in an approximate but conservative way, by composing it with a procedure for deciding termination of depth-bounded systems. Because the respective orderings between types of the two approaches are different in conception, we expect the terminating fragments isolated by the respective systems to be incomparable.

A rather different approach to typing π-terms is presented in [7] where behavioural types are introduced. Roughly speaking, the type system can extract from a π-term P a type which is itself a CCS term simulating P. Properties of the type (such as absence of locks) can then be transferred back to P by virtue of this simulation. By contrast, our types do not carry information about the evolution of the system; if a system is proved depth-bounded by the type system, its evolution can be analysed quite accurately using the decision procedures for depth-bounded systems.

Nested Data Class Memory Automata were introduced [3] as an extension of Class Memory Automata to operate over tree-structured datasets. Without the local acceptance condition, NDCMA have decidable emptiness, and in the deterministic case are closed under all Boolean operations (see [3]). Thanks to these algorithmic properties, NDCMA have recently found applications in algorithmic game semantics [4].

Automata that support name reasoning have been used to model the π-calculus, going back to the pioneering work of History-Dependent Automata [15]. More recently, Tzevelekos [19] introduced Fresh-Register Automata (FRA), which operate on an infinite alphabet of names and use a finite number of registers to process fresh names; crucially it can compare incoming names with previously stored ones. He showed that *finitary π-terms* (i.e. processes that do not grow unboundedly in parallelism) are finitely representable in FRA.

VII. Future Directions

The type system we presented in Section IV is very conservative: the use of simple types, for example, renders the analysis context-insensitive. Although we have kept the system simple so as to focus on the novel aspects, a number of improvements are possible. First, the extension to the polyadic case is straightforward. Second, the type system can be made more precise by using subtyping and polymorphism to refine the analysis of control and data flow. Third, the typing rule for replication introduces a very heavy approximation: when typing a subterm, we have no information about which other parts of the term (crucially, which restrictions) may be replicated.

Let us explain the issue through an example. Let $A = \boldsymbol{\tau}.\nu b.\boldsymbol{\tau}.\nu c.(\overline{a}\langle c\rangle + a(x).\overline{b}\langle x\rangle)^*$ and consider the two terms $P_1 = \nu a.A$ and $P_2 = \nu a.A^*$. The typing derivations for the two terms are almost identical and the set of constraints they impose on $\mathcal{T}$ is the same. However P_1 is depth bounded, P_2 is not. Therefore the type system must reject both. We briefly sketch a possible enhancement that is sensitive to replication. Take the term $\nu b : t_b[t].\nu l : t_l[t].\nu r : t_r[t].b(x).l(y).(\overline{r}\langle x\rangle \parallel \overline{b}\langle x\rangle)^*$ which acts as a 1 cell buffer between l and r. This term cannot be typed by the current type system because $l(y).(\overline{r}\langle x\rangle \parallel \overline{b}\langle x\rangle)$ is migratable for the input $b(x)$ thus requiring $t_l < t_b$, but at the same time $\overline{b}\langle y\rangle$ is migratable for $l(y)$ requiring $t_b < t_l$, leading to contradiction. We propose to add to the structure of $\mathcal{T}$ a notion of multiplicities of base types; a base type can be marked with either 1 or ω. Suppose the forest of a term has a path p from a node n to a node n' where the trace of p consists only of base types marked with 1. This situation will represent the fact that no branching will ever occur between the two replications corresponding to n and n' and having one of the two names in the scope guarantees that the other one is in the scope too. In other words, all the restrictions represented by nodes in p can be though as a indivisible unit; when typing an input term on a name with base type t, the constraints of rule In can be relaxed to require the free variables of migratable terms to have base types smaller than the lowest t' such that the path between t and t' in $\mathcal{T}$ is formed only of base types with multiplicity 1. In the case of buffer example, we observe that b, l and r could all be assigned base types of multiplicity 1 thus replacing the two conflicting constraints with the constraints $t_l \leq t'$ and $t_b \leq t'$ where t' is the greatest among t_l, t_r and t_b. The formalisation and validation of this extension is a topic of ongoing research.

Acknowledgement

We would like to thank Damien Zufferey for helpful discussions on the nature of depth boundedness.

References

[1] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *IFP*, 8(3):121–123, 1979.

[2] H. Björklund and T. Schwentick. On notions of regularity for data languages. In *FCT*, pages 88–99, 2007.

[3] C. Cotton-Barratt, A. S. Murawski, and C.-H. L. Ong. Weak and nested class memory automata. *CoRR*, abs/1409.1136, 2014. To appear in *LATA 2015*.

[4] C. Cotton-Barratt, D. Hopkints, A. S. Murawski, and C.-H. L. Ong. Fragments of ML decidable by nested data class memory automata. In *FoSSaCS*, 2015. To appear.

[5] I. Cristescu and D. Hirschkoff. Termination in a π-calculus with subtyping. In *EXPRESS*, 2011.

[6] R. Hüchting, R. Majumdar, and R. Meyer. A theory of name boundedness. In *CONCUR*, 2013.
[7] A. Igarashi and N. Kobayashi. A generic type system for the π-calculus. In *POPL*, pages 128–141, 2001.
[8] R. Meyer. On boundedness in depth in the π-calculus. In *IFIP TCS*, pages 477–489, 2008.
[9] R. Meyer. *Structural stationarity in the π-calculus*. PhD thesis, Carl von Ossietzky University of Oldenburg, 2009.
[10] R. Meyer and R. Gorrieri. On the relationship between π-calculus and finite place/transition Petri nets. In *CONCUR*, pages 463–480, 2009.
[11] R. Milner. Functions as processes. *Mathematical structures in Computer Science*, 2(02):119–141, 1992.
[12] R. Milner. *The polyadic pi-calculus: a tutorial*. Springer-Verlag, 1993.
[13] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
[14] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I, II. *Inf. Comput.*, 100(1):1–77, 1992.
[15] U. Montanari and M. Pistore. An introduction to history dependent automata. *ENTCS*, 10:170–188, 1997.
[16] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *LICS*, pages 376–385, 1993.
[17] B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *J. ACM*, 47(3):531–584, 2000.
[18] R. M. Reiner Hüchting and R. Meyer. Bounds on mobility. In *CONCUR*, pages 357–371, 2014.
[19] N. Tzevelekos. Fresh-register automata. In *POPL*, pages 295–306, 2011.
[20] T. Wies, D. Zufferey, and T. Henzinger. Forward analysis of depth-bounded processes. In *FoSSaCS*, pages 94–108, 2010.
[21] D. Zufferey, T. Wies, and T. Henzinger. Ideal abstractions for well-structured transition systems. In *VMCAI*, pages 445–460, 2012.